本科毕业论文（设计）

基于区域匹配技术的软件供应链成分分析工具设计与实现

DESIGN AND IMPLEMENTATION OF SOFTWARE SUPPLY CHAIN COMPONENT ANALYSIS TOOL BASED ON AREA MATCHING TECHNOLOGY

丁元川

哈尔滨工业大学

2024 年 5 月

密级：公开

本科毕业论文（设计）

基于区域匹配技术的软件供应链成分分析工具设计与实现

本　　科　　生：	丁元川
学　　号：	120L012424
指　导　教　师：	刘延芳　研究员 李红　副研究员
专　　业：	飞行器环境与生命保障工程
学　　院：	航天学院
答　辩　日　期：	2024 年 5 月
学　　校：	哈尔滨工业大学

摘　要

软件供应链是软件开发过程中形成的复杂网络状供应链结构，开源软件是软件供应链的重要组成部分，开源软件的广泛使用提高了软件开发的效率，但也将漏洞引入了软件供应链。软件供应链成分分析技术能够检测软件中的开源组件成分，是供应链漏洞修复的重要辅助手段。

目前，有许多主流的检测框架可以实现软件成分分析，如 Centris、TPLite 等，但是它们大部分都直接使用库粒度特征或者函数粒度特征进行匹配，难以检测复杂重用关系，存在误报、漏报等问题。针对这些问题，本文提出了一个基于区域匹配技术的框架：LibDetective。LibDetective 把区域匹配技术引入基于源代码的软件成分分析任务，并提出一种基于节点度的图嵌入算法，在区域粒度进行特征匹配，从而更准确地完成了软件成分分析任务。同时，本文以 LibDetective 算法为基础，设计和实现了一款软件成分分析工具。本文对之前工作中的公开数据集进行了调整以适用于源码成分分析的场景，并对 LibDetective 和 Centris 框架进行了测试。结果表明，LibDetective 在两个以复杂重用关系为主的数据集上准确率相较于 Centris 分别提高了 47%和 33%，证明了 LibDetective 性能的优越性。

关键词:软件供应链；开源软件库；软件安全；软件开发；漏洞

Abstract

The widespread use of open source software improves the efficiency of software development, but also introduces vulnerabilities into the software supply chain. Software supply chain component analysis technology can detect the components of open source components in software, which is an important auxiliary means for supply chain vulnerability repair.

At present, there are many mainstream detection frameworks that can implement software component analysis, such as Centris and TPLite, but most of them directly use library granularity features or function granularity features for matching, which is difficult to detect complex reuse relationships, and there are problems such as false positives and false negatives. To solve these problems, this paper proposes a framework based on region matching technology: LibDetective. LibDetective introduces the region matching technology into the software component analysis task based on source code, and proposes a graph embedding algorithm based on node degree to perform feature matching in region granularity, so as to complete the software component analysis task more accurately. At the same time, based on the LibDetective algorithm, this paper designs and implements a software composition analysis tool. In this paper, we adapt the public dataset from the previous work to the scenario of source code composition analysis, and test the LibDetective and Centris frameworks. The results show that the accuracy of LibDetective is 47% and 33% higher than that of Centris on two datasets dominated by complex reuse relationships, respectively, which proves the superiority of LibDetective's performance.

Keywords: software supply chain, open-source software libraries, software security, software development, vulnerability

目　录

第1章 绪 论

1.1 研究背景

现代社会的正常运转离不开软件，软件不仅是信息技术行业的核心，同时在制造业、金融服务等行业都有广泛应用和重要地位。软件是指由一系列计算机程序和相关数据组成的电子化产品[1]，随着硬件技术和网络技术的进步，现代软件朝着大规模化的趋势发展，对于软件开发商而言，完全独立开发软件变得越来越困难。为了提高软件开发的效率，降低软件开发的成本，软件开发者在进行软件开发时会优先选用免费的开源软件。在此背景下，开源软件得到了广泛应用，开源软件产业迅速发展，以 Github、Maven 为代表的开源软件托管平台和开源社区开始如雨后春笋般出现，开源社区的注册用户数量和托管软件数量逐年增长。在软件开发的过程中，开源软件会更多的作为目标软件的组件，实现目标软件的部分功能，软件重用过程中形成的复杂网络状结构称为软件供应链。开源软件为软件开发带来好处的同时，也将潜在的漏洞引入软件供应链，由于开源软件本身的漏洞导致下游软件产生安全问题的案例数不胜数，给个人、企业和国家造成了损失和威胁。

在这种背景下，软件成分分析技术应运而生。软件成分分析技术通过对目标软件的源代码或二进制进行分析，从而判断目标软件重用的开源软件类型、版本等信息。解析出开源软件的重用情况后与漏洞数据库进行关联，即可判断目标软件潜在的漏洞。目前主流的软件成分分析算法主要处理输入为源代码的情况或输入为二进制的情况，匹配粒度上主要基于库级粒度或函数级粒度。由于重用粒度与区域粒度不匹配，库级粒度和函数级粒度均会产生漏报和误报问题，降低检测的准确度。最近的一项学术成果提出了区域粒度，并验证了在输入为二进制的情况下区域粒度与库级粒度和函数级粒度相比检测效果得到显著提升，但基于区域粒度且输入为源代码的研究存在空白，暂时没有相关成果。

因此，本文以软件成分分析为目标，通过将区域粒度引入输入为源代码的软件组成分析任务并提出和应用了一种基于节点的度的图嵌入算法，设计和实现了一种软件成分分析算法，并以该算法为内核开发了一款软件成分分析工具。同时利用本文开发的工具与相同应用场景的主流算法在相同数据集上进行对比实验，以检测本文设计算法的准确性。

1.2 国内外研究现状

1.2.1 开源软件研究现状

开源软件是指使用共享源代码、开放标准以及允许全球软件开发人员和用户协作构建、纠错以及增强功能的软件[2]。与传统软件开发的形式不同，用户可以免费获得开源软件的源代码，并对其进行利用或改造，改造包括改错、移植和建立附加程序等[3]。

开源软件具有悠久的历史，早在计算机出现的时期[4]，开源软件就已经产生。开源软件在 20 世纪 60 年代末[5]就已经很普遍，当时网络技术还不发达，软件仅在软件管理员之间传递，在此过程中，软件的源代码也在不断地被传递，例如，当时的 UNIX 系统就是提供源代码的[6]。在后续的发展中，不断有新的开源软件的诞生。

与开源软件相对应的是闭源软件，闭源软件与开源软件的最大不同就是闭源软件的源代码不能被用户直接接触到。开源软件与闭源软件相比，尽管存在质量低下等问题[7]，但是开源软件具有免费、可扩展性强等优点[7]，这些优点促使了开源软件的频繁使用和开源软件市场的蓬勃发展。

开源软件产业发展迅速，项目数量和用户数量都在逐年增长。以 Github 为例，截至 2022 年 6 月，GitHub 已经有超过 5700 万注册用户[8]和 1.9 亿代码库（包括至少 2800 万开源代码库）[9]，使其成为了世界上最大的代码托管网站和开源社区[10]。2023 年 1 月 26 日，GitHub 已经被超过 1 亿开发人员使用[11]。

开源软件的流行并不是偶然，而是市场发展规律决定的必然，它的影响范围注定会超出软件行业本身。早在 2003 年，Bonaccorsi [12]等人就已经研究了开源软件的出现和传播所引发的三个关键经济问题：动机、协调和在主导标准下的传播，从经济学的角度得出了开源软件必将繁荣发展的结论。在 21 年后的今天，开源软件已经成为社会计算机行业的重要组成部分，而且它的影响已经超出了计算机行业本身。开源软件在软件领域、教育领域[12]、航天领域[13]等都有着直接的应用。随着网络技术和硬件技术的发展，未来软件也向着高度复杂化的趋势发展，软件开发者会更多的选用现成的组件进行新软件的开发，开源软件会更多的作为大型软件的组件，得到更为广泛的应用。

1.2.2 软件供应链和软件供应链安全问题研究现状

近年来，开源软件快速发展，已成为现代信息产业至关重要的一环，极大的改变了软件的构成形式，给开发过程带来了巨大变化[14]。软件开发者和软件制造商为提高效率、降低成本，会优先选择使用相关第三方开源组件，并在此基础上进行

修改、拓展和移植，在这个过程中形成新的软件开发方式称为开源协作模式[15]。在开源协作模式下，软件功能更多样化，架构更规模化，形成了更加复杂的网络状的供应链结构[14]，这条网络状的供应链被称为软件供应链。软件供应链由用于开发、构建和发布软件产品的组件、库、工具和流程组成[15]。如图 1-1 所示，软件供应链包含许多环节，每个环节还可以细分，软件供应链的上下游相互依赖，互相影响。

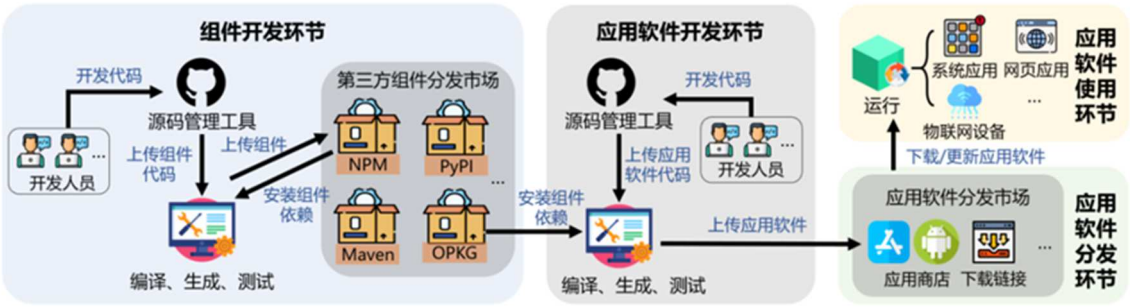


图 1-1　软件供应链示意图[14]

随着软件供应链的出现，第三方库的概念也随之产生。第三方库也可以被称为第三方软件组件[16]，它指的是在软件开发过程中所重用的软件组件，第三方软件可以是开源免费的，也可以是付费的[16]。

软件供应链为在软件开发带来便利的同时，也存在安全问题，导致问题的主要原因是开源软件的广泛使用引入的安全隐患[18]。开源软件是软件供应链中的重要组成部分，而软件供应链中的任何一环的参与者，无论是出于意外还是恶意，都可能破坏下游软件[19][20]。软件供应链中存在的安全问题已经导致了大量网站的破坏和全球互联网范围内服务的中断，损失超过数十亿美元[21][22]，这些问题包括服务中断[23][24]和危及人类生命[25]与国家安全[26]的网络安全漏洞。软件供应链的漏洞可能归因于与需求不匹配——软件供应链最初被设计用于共享，而非网络安全[27]，对软件供应链的改进和完善需要时间与经验，因此软件供应链会长期面临着被攻击的风险。

国际上，Synopsys 发布的《2022 年开源安全和风险分析报告》显示，97%的软件代码重复使用了开源代码作为组件，甚至包括使用多年前的未修补的开源代码，81%的软件存在至少一个已知安全漏洞，而开源组件平均每天会披露 200 多个新漏洞，并且物联网、航天和互联网行业的安全风险尤为突出[28]。在国内，奇安信公司发布的《2021-2023 中国软件供应链安全分析报告》显示，在被分析的 2631 个国内企业软件项目中，全部都使用了开源软件，使用率为 100%，最多的项目使用了 5695 个开源软件，平均使用了 155 个，由于开源软件的使用，平均每个软件项目存在 110 个已知开源软件漏洞，远高于前两年的 69 个和 66 个[29]。国内外近几

年的软件供应链安全报告都表明软件供应链的安全形势不容乐观。2014 年 4 月，心脏滴血漏洞[30]突袭了互联网。这个漏洞是自商用互联网诞生以来最为重大的漏洞之一，估计有 24%至 55%的热门 HTTPS 网站受到影响。2021 年 10 月，人类互联网史上最严重的漏洞之一 log4j 漏洞[30]袭击了互联网。log4j 是 Java 语言的日志组件，log4j 附带漏洞，导致 90%以上基于 java 开发的应用平台都受到影响。图 1-2 展示了开源软件影响软件供应链安全的原理，下游软件重用了上游软件作为组件，组件中包含的漏洞也随着供应链流动到下游，下游所有使用该组件的软件都会被污染。上面两个典型事件均是因为下游软件重用不安全的上游开源组件造成的，像这样由于开源软件的不安全性导致软件供应链面临威胁的事件很多，给用户和企业甚至是国家带来了巨大的损失和严重的威胁。

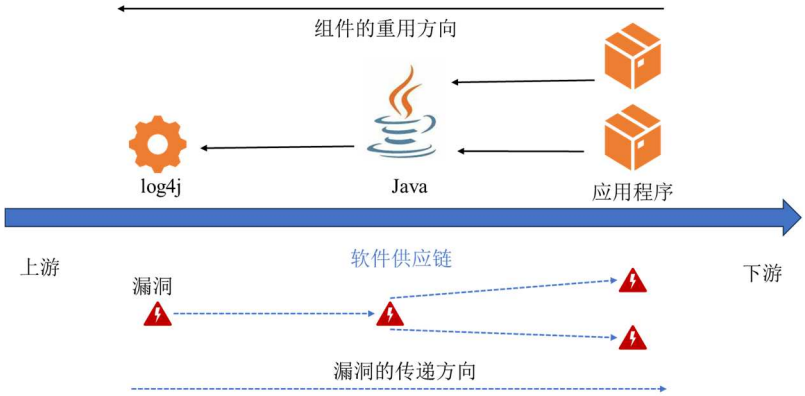


图 1-2　软件供应链上下游组件与漏洞的传递方向示意图

为了提高软件供应链的安全性，美国国家安全系统委员会（CNSS）于 2022 年 8 月发布了给软件开发者的软件开发手册[32]，从政府层面上提出应对软件供应链安全问题的措施。学术界和产业界也不断提出各种保障软件供应链安全的算法和工具。无论是现在还是将来，软件供应链安全都会是网络安全领域的一个重要研究方向。

1.2.3　航空航天产业与软件供应链安全问题关系研究

开源软件在航空航天产业上得到了广泛的应用，并且产生了深刻的影响。早在 2014 年，德国宇航中心的工作人员就已经注意到了开源软件的显著优点，并开始广泛使用开源软件[33]。他们指出，软件开发是德国航空航天中心（DLR）大多数研究所的核心任务，大约有四分之一的德国航空航天中心人力被分配到软件开发项目中。在大多数软件开发项目中，持续使用免费提供的开源软件可以显著减少开发时间。有时候，一个项目所需的软件只有 10%需要从头开始编写[33]。美国作为航空航天强国，很早就在航天产业中大量使用了开源软件，并且取得了丰硕的成果。

美国国家航空航天局（NASA）在其官网上推出了超过一百种开源软件[34]，这些开源软件基于 C++、JavaScript 等语言编写，可以用于仿真、软件开发等。2020 年，美国 SpaceX 公司星链计划的主管 M.Monson 表示， SpaceX 使用了 Linux 作为火箭和卫星的操作系统，他们星链计划已经将 32000 台 Linux 计算机送到太空中[35]。SpaceX 在龙飞船及其火箭（猎鹰 9 号和星际飞船）的主要飞行计算机上使用 3.2 版本的 Linux 内核，飞行软件使用 C++[36]。类似的，NASA 在开发飞行软件的过程中也利用了大量的 C++开源软件[36]。NASA 也广泛的使用了 Linux 系统，例如，NASA 使用的某些截波器在骁龙 801 处理器上使用了某些版本的 Linux 内核。

随着人工智能技术的发展，人工智能技术在各行各业的应用也越来越广泛，在航空航天产业中也不例外。例如，计算机视觉技术在航空航天产业中发展迅速，计算机视觉正在通过提高安全性和效率来改变航空航天产业，它可以自动执行危险检测并简化生产，有着巨大的应用前景[37]。除了操作系统和飞行软件包含的开源组件外，随着人工智能技术的引入，航空航天产业也用到了越来越多人工智能方面的开源软件，如 TensorFlow、Pytorch 等。

遗憾的是，在航空航天产业中常用的开源软件在安全性上面临着严重的威胁。根据奇安信公司发布的《2021-2023 中国软件供应链安全分析报告》，截止 2022 年底，在大型开源项目中，Linux 内核的历史漏洞总数达到了 5337 个，连续 3 年第一，且 Linux 内核在 2022 年一年间漏洞增量达到了 579 个，也是第一。在主流软件包生态系统中，TensorFlow 框架的历史漏洞数达到了 410 个，比第二多了 89 个，TensorFlow 在 2022 年一年间新增漏洞数达到了 165 个，比第二多了 121 个[29]。

软件成分分析技术通过对航天软件进行扫描分析，可以发现软件中潜在的漏洞，辅助漏洞关联和漏洞修复，为航空航天产业的软件安全保驾护航。

1.2.4 软件组成分析技术研究现状

软件组成分析是信息技术和软件工程领域的一种技术，用于分析定制的软件应用程序所嵌入的开源软件成分，并检测它们是否是最新的、是否包含安全缺陷或是否具有许可要求[38]。软件组成分析也可以理解为分析开发的软件重用了哪些第三方库[17]。由于付费软件需要付费的性质，进行检测时数据库中付费软件的内容很少，因此软件组成分析主要面向免费的开源软件，在这种情况下，第三方库近似等价于开源软件。软件组成分析技术是应用程序安全测试（AST）工具市场的一部分，可用于管理开源组件。软件组成分析工具对应用程序的代码库（包括容器和注册表等相关部分）进行自动化扫描，识别出软件或其组件的许可证合规性数据、开源组件和所有安全漏洞。除了能使开源使用可视化，一些软件组成分析工具还可以

通过划分优先级和自动修复来辅助修复开源漏洞。

软件组成分析是一个新兴行业，虽然只有 20 多年的历史，但是发展迅速，已经经过了 5 个阶段。2002 年左右，第一款开源手动扫描仪发布[39]。尽管它能提高企业代码库的可见性，但这种早期技术的误报率很高，需要人工介入排查和筛选得到的数据，无法处理高要求的任务场景。到 2011 年，该技术已得到改进，实现了自动化的漏洞和许可实时监测，这使得软件和安全团队能够在开发周期的早期部署开源管理[38]。再后来，软件组成分析解决方案还与存储库、构建工具、包管理器和 CI 服务器等软件开发工具集成[39]。据估计，一些组织多达 90%[39]的源代码现在都是开源的，这意味着潜在的攻击面已经并将继续扩大。这增加了开源漏洞的风险，以及恶意软件包渗透、攻击软件和应用程序的威胁。因此，软件组成分析工具变得越来越重要。在需求的推进下，软件组成分析工具将变得越来越智能化。

软件组成分析产品除了具有自动化的特性，它还有提高软件安全性、节约成本、提高开发和维护效率的优势，它的存在使得开发人员在使用和集成开源软件组件时不必手动进行额外的工作[40]。这种自动化也适用于代码和构件中对其他开源软件组件的间接引用。

同时，软件组成分析产品也存在着一些局限性。例如，每个公司的产品都使用其专有的数据库，这些数据库在大小和覆盖范围上可能存在显著差异。另外，组成分析得到的漏洞数据仅是报告在美国国家漏洞数据库官方中的漏洞，如果漏洞数据库未更新，则其不能关联到最新的漏洞[41]。

总而言之，软件组成分析是一种查找漏洞的方法，并不是一种挖掘新漏洞的方法。它专注于识别和修复开源组件和第三方依赖项中的风险，并不能挖掘开源软件代码或者自定义代码中的漏洞。

1.2.5 软件组成分析算法研究现状

软件组成分析任务本质上是分析软件中第三方库的重用情况，因此也可以被称为第三方库重用检测任务。鉴于第三方库的重要性和第三方库重用现象的普遍性，第三方库检测这一课题在五年内得到了迅速发展。无论是在学术界还是产业界，都取得了长足的进展，产生了一批优秀的学术成果和产品，但是由于课题的复杂性，暂时还没有能完美检测的算法和产品出现。

第三方库之间的重用关系的影响检测效果的重要因素。检测根据 Li[42]等人的研究，第三方库的重用形式可以分为全部重用、部分重用、嵌套重用和伪传播重用 4 种方式。全部重用和部分重用是较为简单且直接的重用方式，较容易被检测出来；嵌套重用和伪传播重用较为复杂，更难被检测出来。Centris[43]和 TPLite[44]等框架

并不能检测复杂的重用关系，最近提出的一些新的框架能够有效地改善这一问题，但是也并不能完全准确的检测出所有的复杂重用关系。

现有第三方库重用检测方法根据检测特征的粒度分为库级粒度特征和函数级粒度特征。基于库级粒度特征的检测方式旨在提取整个第三方库的特征，通过比较库与库之间特征的差异，来判断库之间的重用关系。基于函数级粒度特征的检测方式旨在提取第三方库中每一个函数的特征，通过得出函数与函数之间的相似度，来判断函数之间的重用关系，从而判断库与库之间的重用关系。

库级粒度特征相对比较简单，相关的研究进行更早。2017 年，美国佐治亚理工大学的 R.Duan[45]等人提出了 OSSPolice 框架，该框架基于二进制以检测重用关系。虽然检测效果与之前的框架相比有所提高，但是它不能有效的检测复杂的重用关系，导致存在较高的误报率。同年，中国科学院信息工程研究所的 Feng[46]等人提出了 B2SFinder 框架，实现了开源软件在 COTS 软件（一类开源软件）中的重用检测。但是由于基于库级粒度的这几种方法都有明显的缺点：比较粒度与第三方库的重用粒度不匹配，不能很好的检测复杂重用关系，容易导致漏报问题。

最近的一些研究表明，函数级粒度特征可以获得更精确的结果。同时，适用于不同应用场景且基于函数级粒度特征的检测框架开始出现。在输入为源代码的情况下，代表性的工作是 Centris[43]和 TPLite[44]。2021 年，韩国高丽大学的 Woo[43]等人提出了 Centris 框架，能够很好的检测相对简单的重用关系，但随着现在软件重用关系的复杂化，该框架的局限性也愈发明显。2023 年，浙江大学的 Jiang[44]等人，在 Centris[43]框架的基础上，提出了 TPLite 框架，它对 Centris[43]不合理的地方进行了改良，提高了检测的准确度，但是仍然不能有效地检测复杂重用关系，导致不准确。在输入为二进制的情况下，代表性的工作很多。2022 年，浙江大学的 Zhao[47]等人提出了 FirmSec 框架，实现了固件中第三方库的检测和漏洞分析。清华大学的 Tang[48]等人提出了 LibDB 框架，实现了第三方库的匹配。2023 年，新加坡南洋理工大学的 Wu[49]等人提出了 OSSFP 框架，该方法精确且可拓展，使用指纹功能，实现了对 C/C++第三方库的检测。上面的方法基于函数级粒度，容易产生误报问题，同时，虽然能够处理复杂的重用关系，但是准确度不高。区域匹配技术的引入很好的解决了这些问题。中国科学院信息工程研究所的 Li[50]等人提出了 LibAM 框架，该框架创新性的提出并使用了区域匹配技术，显著地提高了复杂重用关系的准确度, 实现了第三方库重用的检测。

目前，有许多能够进行第三方库检测的产品，如南洋理工大学的 Scanist，Synopsys 公司的 Black Duck。2024 年南方科技大学和腾讯科恩实验室提出了 BinaryAI [51]框架，并推出了同名的产品 BinaryAI。

总而言之，现有的第三方库检测方法很多，它们有着不同的应用场景。从输入上区分，可以分为基于源代码型、基于二进制型或者基于源代码二进制结合型；从应用场景上区分，有面向固件的、面向移动端 APP 的，也有面向普通软件的；从匹配特征的粒度上面区分，可以分为基于库级粒度、基于函数级粒度或基于区域粒度。然而，基于库级粒度或函数级粒度的方法由于检测粒度与实际重用粒度不一致，存在许多漏报和误报。Li[50]等人的工作已经证明了区域匹配技术通过区域粒度的匹配，能够显著提高检测工具性能，但是该工作只局限于二进制级别的成分分析，大量 C/C++第三方库无法大规模自动编译为二进制，需要在源码级别进行成分分析，目前暂时不存在有效的从源码级别进行分析并基于区域粒度的算法和工具。

1.3 本文主要研究内容

鉴于开源软件使用的广泛性和软件组成分析技术的重要性，本文使用区域匹配技术，提出了基于节点的度的一种新的图嵌入算法，设计出了基于源代码的第三方库重用检测工具：LibDetective。本文通过将孤立的函数连成区域，将区域匹配引入输入为源代码的方向来更加准确的检测第三方库；同时提出了基于节点的度的一种新的图嵌入方法，实现了快速匹配函数对从而检测第三方库的效果。本文设计的框架能够有效的过滤现有方法的误报和漏报，在很短的时间内准确检测第三方库，且能够检测到函数级粒度的重用结果。在实现 LibDetective 算法的基础上，本文利用 Apache 软件和 Flask 框架，开发了一款能使用的浏览器/服务器架构的软件，完成了软件组成分析工具的设计与实现任务。在实现了 LibDetective 软件的基础上，为了验证 LibDetective 的性能，本文进行了对比实验。用相同数据集同时测验 Centris[42]和 LibDetective，结果显示 LibDetective 框架的准确度和召回率相比于 Centris[42]有了显著的提升，验证了 LibDetective 框架性能的优越性。具体研究工作如下：

（1）广泛调研了开源软件、软件供应链及其安全问题、软件组成分析技术和软件组成分析算法的研究现状，总结了现有的软件组成分析算法存在的问题，为后续的研究指明思路。

（2）完成软件成分分析算法的设计和实现工作。从数据库的构建到函数特征的提取，再到区域相似度的比较，完整的建立了较为准确的软件成分分析算法，为后续以该算法为核心的软件组成分析工具的设计与开发奠定了基础。

（3）完成软件成分分析工具的设计与开发工作。通过对工具面向的项目需求进行分析，从软件框架的选择到服务器软件和服务器框架的选择，再到软件前后端代码的编写，完整的实现了软件成分分析工具的功能，为后续以该工具为基础的对

比实验的进行奠定了基础。

（4）完成本文搭建的软件成分分析工具与当前主流的软件成分分析工具的对比实验。利用本文开发的软件成分分析工具与相同应用场景下的主流框架在相同的数据集上展开测试，计算准确率、精确率和召回率并进行对比，验证了本文设计的软件成分分析算法的优秀性和软件成分分析工具的实用性。

根据论文的主要内容，本文的章节安排如图 1-3 所示：

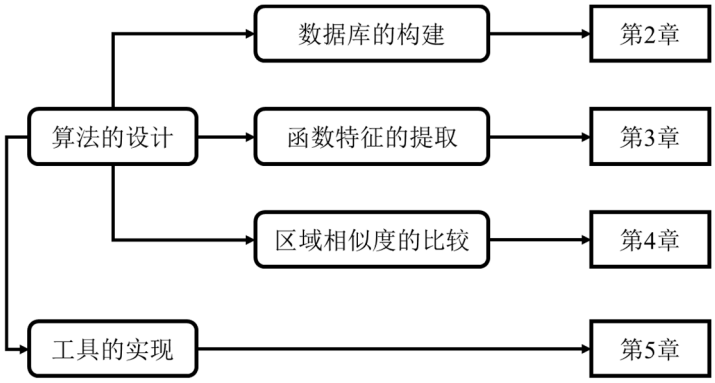


图 1-3　本文工作与章节对应关系

第 2 章至第 4 章为算法设计阶段，算法设计阶段包括数据库的构建、函数特征的提取和区域相似度的比较 3 步。第 2 章为数据库的构建阶段，通过解析源代码文件，将文件中定义的函数保存为哈希值，存储到数据库中，为函数特征特征提取阶段打下了基础。第 3 章为函数特征的提取阶段，通过分析数据库的构建阶段所形成的数据库和获取的源代码，分别提取函数的文本特征和结构特征，引入了区域相似度的比较阶段。第 4 章为区域相似度的比较阶段，通过提取函数的文本特征缩小了匹配范围，对文本特征匹配成功的函数利用其结构特征进行区域相似度的比较，即可判断函数之间的相似关系，从而推断出开源软件和目标软件之间的重用关系。算法设计是工具实现的基础，该算法为本文所设计工具的内核。

第 5 章为工具实现阶段，工具实现阶段以第 2 章至第 4 章实现的算法为内核，实现了一款可用的软件成分分析工具，并利用该工具与目前的主流算法在相同数据集上进行性能测试，以验证工具的准确性。

第 2 章　数据库的构建

2.1　引言

LibDetective 框架引入区域粒度，通过比较目标软件与开源软件之间函数的相似度来判断目标软件与开源软件之间的重用关系，因此，解析出目标软件和开源软件中定义的所有函数是实现算法的第一步。构建数据库的本质，就是解析开源软件源代码中的所有文件，得到文件中定义的所有函数，并以合理的算法处理和存储函数，将结果保存到对应的数据库中。

随着科学技术的发展，网络和硬件的性能得到极大的提升，促进现代软件向大规模化发展，应用市场上的开源软件也随之变得规模大，版本多。单个开源软件中定义函数的数量是庞大的，少则数千个，多则达数千万个，这导致进行第三方库检测时，如果将每种库的每一个版本单独存储为一个数据库，不仅会占用很大的空间，而且由于同一种软件的不同版本之间存在大量相同或相似的函数，如果直接将未处理的大量数据库输入后续阶段，会导致不同版本中相同的或相似的函数被匹配多次，从而导致不必要的时间开销，降低后续阶段的工作效率。因此，有必要利用基于开源软件各个版本构建的数据库进行消除冗余操作，构建最终数据库，将不同版本中相同的或相似的函数信息合并到最终数据库的同一位置，使每种开源软件只对应一个数据库。经过消除冗余操作后获得的精简数据库显著地节省了内存空间，也有效地降低了后续阶段的时间开销。

在这种背景下，数据库构建阶段需要完成创建原生数据库、消除冗余得到精简数据库两个步骤，具体工作流程如图 2-1 所示。本章需要从代码托管平台上获取指

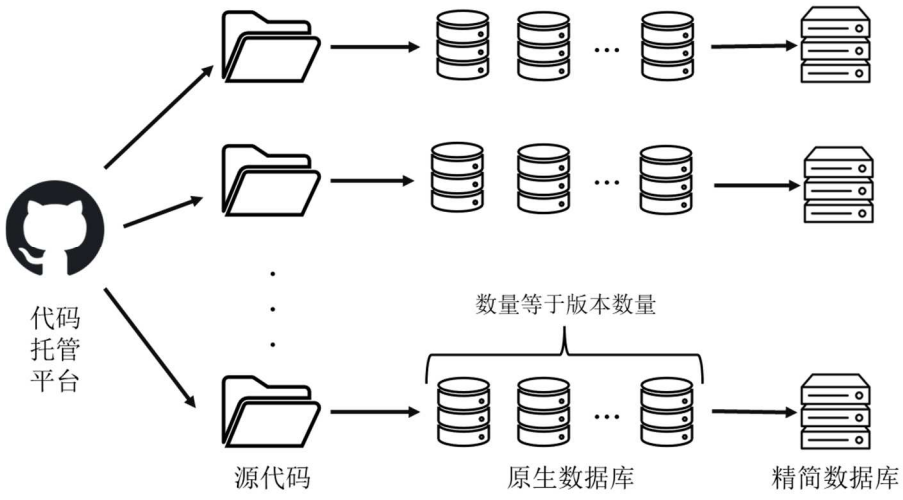


图 2-1　数据库构建阶段工作流程

定开源软件的源代码，再对源代码进行分析和处理，得到开源软件的原生数据库。开源软件原生数据库的数量等于该软件版本的数量，每一个版本对应一个原生数据库。对所有原生数据库进行消除冗余操作，得到精简数据库，每一种开源软件只有一个精简数据库。精简数据库是第 3 章中函数特征提取阶段的输入，本章的最终目的是得到每种开源软件对应的精简数据库。

本章设计了合理的算法，选择了合适的工具，实现了数据库的构建任务，为后续阶段奠定了基础。

2.2 原生数据库的构建

2.2.1 源代码的获取

Github 是世界上最大的开源社区和代码托管平台，因此本文主要获取 Github 平台上的开源软件。Github 基于 Git，可以使用克隆命令将需要的文件从服务器下载至本地。本文需要下载多种开源软件的源代码，对于每种开源软件都需要下载各个版本的源代码。因此，需要设计一个程序脚本，自动化地对文件的源代码进行下载。从服务器上获取源代码的流程图如图 2-2 所示，程序读取和分析文本，构造克隆指令，然后调用本机的 Git 软件，执行克隆命令，最后将从服务器返回的源代码存储到相应位置。在下载文件时可能受到网速等因素的影响，从而导致无法获取某

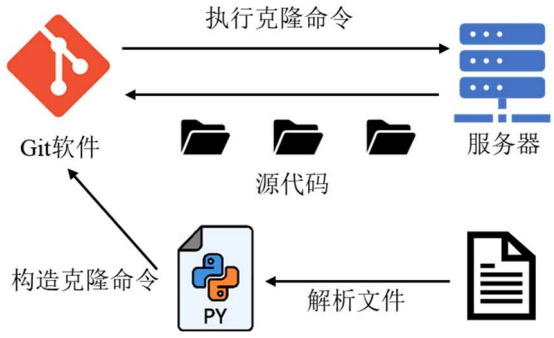


图 2-2 通过构造克隆命令从服务器上获得源代码

些库的源代码，因此要做好异常处理，让程序不会停止运行。源代码获取的具体实现算法如表 2-1 所示。

表 2-1 源代码获取

算法：源代码文件的自动化获取
输入：包含需要下载的第三方库源代码链接的文件
输出：保存到指定位置的源代码文件
主函数 main：
try：

表 2-1（续表）

算法：源代码文件的自动化获取
输入：包含需要下载的第三方库源代码链接的文件
输出：保存到指定位置的源代码文件
读取文件
catch：
说明异常原因
try：
解析克隆命令
下载文件到指定路径
catch：
说明解析错误库的名称，说明异常原因

源代码获取算法的实现基于 python 语言，并利用 python 的 subprocess 库调用了 Git 软件。subprocess 库是 python 标准库中的一个模块，python 程序脚本能调用该库创建新的进程，并连接到新创建进程的输入、输出和错误管道，并且可以获取进程的返回值。subprocess 库使用 run 函数开始运行进程该函数能执行指令并等待进程结束，最后返回一个 CompletedProcess 实例，指令执行返回的结果是 CompletedProcess 实例的一个属性，解析该实例即可得到需要的结果。Git 软件利用固定格式的指令从互联网上获取指定资源，因此可以通过构造指令来自动化的批量下载源代码。以本文场景为例，下载某个开源软件源代码的指令格式如图 2-3 所示，指令格式等于固定前缀、目标载荷和固定后缀的连接，固定前缀的内容是“git clone”两个单词，固定后缀的内容是“.git”后缀，指令格式的固定前缀部分用空格符与其他两部分连接。以下载 lzfse 软件为例，该软件源代码的链接为 https://github.com/lzfse/lzfse，命令格式为 git clone https://github.com/lzfse/lzfse.git。

(git clone) + (开源软件源代码的网址链接) + (.git)

固定前缀　　目标载荷　　固定后缀

示例

git clone https://github.com/lzfse/lzfse.git

图 2-3　Git 软件下载指定资源的格式和示例

源代码获取算法的流程如图 2-4 所示，程序首先读取写有目标开源软件源代码网址链接的文本文件，将文本文件中的条目作为目标载荷，与固定前缀和固定后缀进行连接形成指令，然后使用 subprocess 库的 run 函数执行指令，解析 run 函数返回的 CompletedProcess 实例，即可得到开源软件的源代码，将源代码按照开源软件

的类型分别存储在不同的文件路径下。解析完文本文件中所有的条目，即可实现源代码的获取任务。

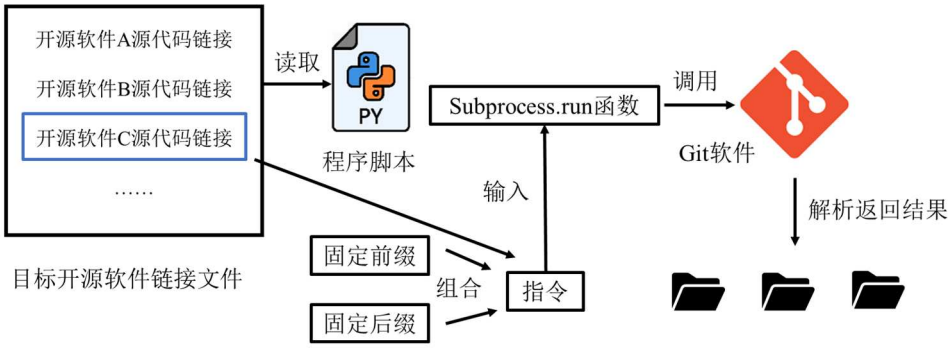


图 2-4　源代码的获取具体流程图

2.2.2　源代码的预处理

鉴于不同的软件开发者有不同的编程习惯，获取到的源代码会有不同的风格和格式，这使得源代码包含了很多生成数据库时不需要的信息，例如注释、换行符、制表符等内容。同时，软件开发者在利用开源软件时，常规的操作是在源代码的基础上添加注释、换行符等信息，这导致相同开源软件的源代码在不同的目标软件中存在内容上的差异。这些不必要的字符串导致功能相同的函数在文本上表现得不同，使相同的函数经过局部敏感哈希算法的处理后得到不同的局部敏感哈希值，从而影响生成原生数据库阶段的准确率。因此，在生成数据库之前，需要对源代码进行预处理，以提高数据库中元素的准确度。如图 2-5 所示，通过去除注释、制表符和换行符等无关字符串，预处理阶段屏蔽了开源软件开发者和开源软件使用者对开源软件源代码信息的添加，将函数本质的信息输入局部敏感哈希函数[52]，保证了不同软件中使用的相同开源软件能被判别为相同，提高检测的准确率。

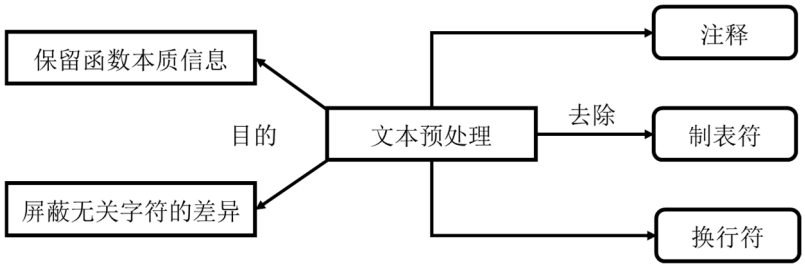


图 2-5　文本预处理阶段的目的和去除对象

对文本进行预处理消除指定内容的方法很多，正则表达式是一种快速匹配字符串的方法，它是一种强大的文本处理工具，可以用来匹配、搜索、替换文本中的特定模式。因此通过构造合适的正则表达式对函数内容进行匹配，将匹配到的字符串替换为空，可以完成对指定内容的删除。

文本预处理算法基于 python 语言，调用了用于处理 python 正则表达式的 re 库，具体流程如图 2-6 所示。re 库提供了 compile 函数以编译正则表达式模式，并返回一个可重用的正则表达式模式，然后利用 re 的 findall 函数寻找匹配到的所有满足该正则表达式的字符串，将这些字符串替换为空，即可完成文本预处理。

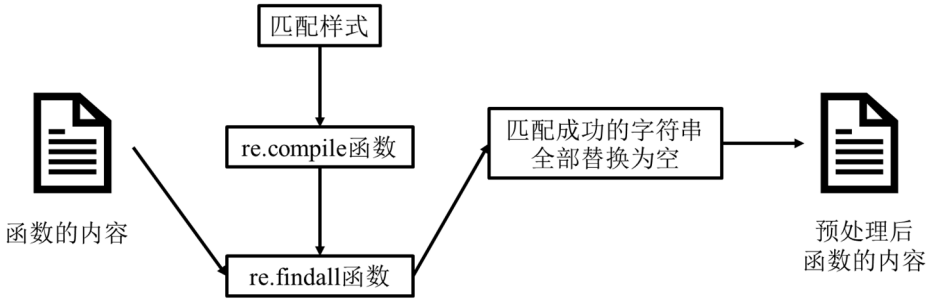


图 2-6　文本预处理的具体实现

2.2.3　原生数据库的生成

经过预处理后的源代码没有了多余的内容，只保留了源代码中最基本的信息。在此基础上，利用 ctags 工具对预处理后的源代码进行静态分析，可以得到文件中函数的名称和函数的内容等信息。ctags 工具与目前主流的静态分析工具 Coverity 的对比如表 2-2 所示，ctags 工具是一款对 C 语言、C++程序进行静态分析的开源

表 2-2　ctags 与主流静态分析工具 Coverity 的对比

	ctags	Coverity
规模	小	大
运行速度	快	快
功能全面性	差	优
免费或付费	免费	付费

免费工具，尽管它功能不够强大，但在处理相对简单的任务场景时，它仍具有较高的准确率和较快的解析速度。越相似的文本，经过局部敏感哈希函数[52]处理后，其局部敏感哈希值就越接近，局部敏感哈希函数[52]的这一特性使得其被广泛的应用于文本相似度的比较任务。利用局部敏感哈希函数[52]将源代码中各个函数的内容分别映射为一个定长的局部敏感哈希值，该哈希值是对函数文本特征映射的结果，因此原生数据库只需要保存定常的函数局部敏感哈希值，而不用保存冗杂的函数具体内容。利用局部敏感哈希值表示函数特征并保存到原生数据库中，能显著的减少内存的占用，也是后续通过比较函数局部敏感哈希值来判断函数相似性，从而消除冗余构建精简数据库的基础。生成源代码中所定义函数的局部敏感哈希值的过程如图 2-7 所示。为了让后续步骤更加精确，局部敏感哈希值的长度应尽量长，本文选择哈希值的长度为 72 位，每一位代表一个十六进制的数，取值范围为 0 到 F。

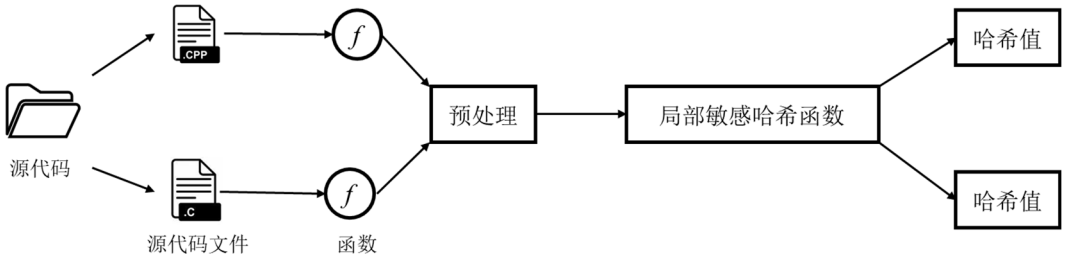


图 2-7　函数局部敏感哈希值产生过程

产生的原生数据库如图 2-8 所示，一种开源软件的每一个版本对应一个数据库，数据库中存储着函数名字、定义函数的文件在原项目中的路径和函数内容的局部敏感哈希值三种信息。

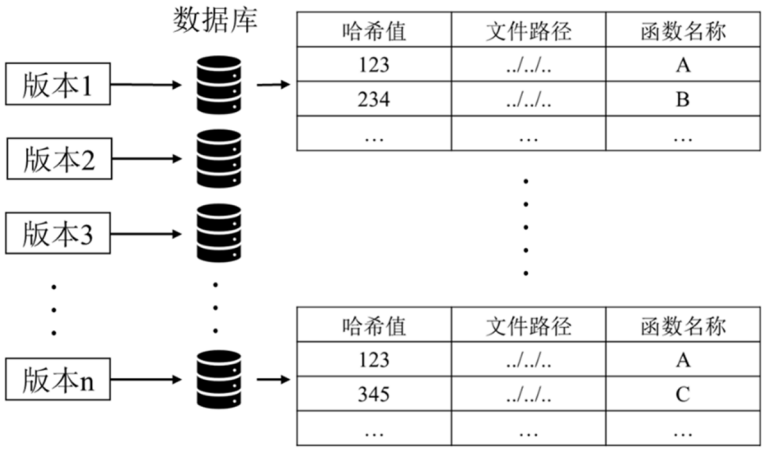


图 2-8　原生数据库示意图

生成目标开源软件原生数据库的算法如算法表 2-3 所示，该算法通过遍历开源软件源代码，调用 ctags 工具解析源代码中目标软件获得函数信息，再使用正则表达式进行文本预处理、局部敏感哈希函数提取函数特征，最后将函数特征、函数名称等信息存储到数据库中。

表 2-3　生成目标开源软件原生数据库的算法

目标：生成目标开源软件的原生数据库
输入：目标开源软件的各个版本的源代码
输出：目标开源软件的所有原生数据库
主函数 main：
for（遍历存储开源软件源代码的文件夹）：
获取开源软件名称和版本号
新建数据库，初始化为空
for（遍历该版本开源软件源代码中的所有文件）：
if（文件的后缀名是.c、.cpp 或.cc）：
构造解析指令

表 2-3（续表）

目标：生成目标开源软件的原生数据库
输入：目标开源软件的各个版本的源代码
输出：目标开源软件的所有原生数据库
调用 ctags 工具解析目标文件
try：
解析文件
分析解析结果，获取文件中定义的函数名称和函数内容
利用构造的正则表达式对函数内容进行预处理
利用局部敏感哈希函数处理预处理后的文本
生成函数的局部敏感哈希值
新建条目
向条目中写入函数名称，文件路径，局部敏感哈希值 3 部分内容
向数据库中添加条目
catch：
输出异常原因和解析错误的文件路径
保存数据库，命名为（开源软件名称+版本号.hidx）

生成原生数据库算法基于 python 语言，主要利用了 tlsh 库，tlsh 库是 python 环境中专门用于计算局部敏感哈希值的模块。tlsh 库提供了 hash 函数，能将英文文本映射为定常的哈希值，通过对所有函数内容的映射，得到所有函数的局部敏感哈希值，就能构建原生数据库。

2.3 精简数据库的构建

由于网络与硬件技术的发展，软件功能也越来越强大，软件的版本也越来越多。以 OpenCV 库为例，OpenCV 目前最新版本为 4.9.0 版，历史发行版本超过 50 个。如果不对数据库进行消除冗余操作，单是 OpenCV 库，就会生成超过 50 个数据库，而且这些数据库之间存在大量相似或相同的内容。冗余的数据库会占用大量的空间，也会显著降低后续阶段的工作效率。因此，对数据库进行消除冗余操作，将所有版本合并到一个数据库，可以节省大量空间，同时为后面的匹配操作节省大量时间。

首先创建一个新的空数据库，消除冗余操作可以将许多版本中都出现的功能相同或者相似的函数合并到该数据库的同一个位置，同时也将每一个新出现的函数写入该数据库的新位置。函数相似度的计算利用差异长度算法实现，该算法是比较字符串相似度的算法。在数据库中，函数特征是函数的局部敏感哈希值，并且由于局部敏感哈希函数[52]越相似的文本产生的局部敏感哈希值越接近的特性，可以

将函数的相似度比较任务转化为函数局部敏感哈希值字符串粒度差异比较任务，此时选择差异长度算法作为局部敏感哈希值差异的计算算法是合理的。差异长度算法能计算一对函数局部敏感哈希值之间的相似度得分，将相似度得分与设定的阈值相比较，即可判断函数之间的相似度，工作流程如图 2-9 所示。精简数据库的

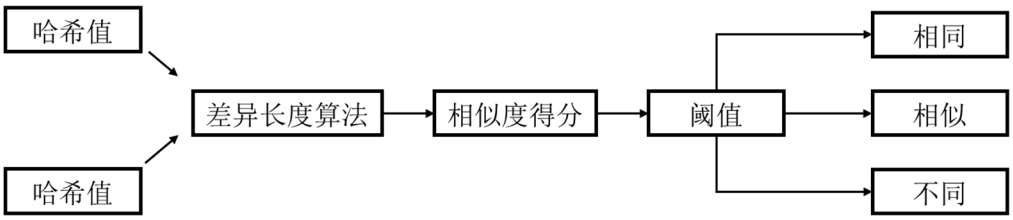


图 2-9　相似度判断流程图

生成过程如图 2-10 所示，遍历一种开源所有版本的原生数据库，同时进行消除冗余操作，即可得到精简数据库。精简数据库中存储了函数名字、定义函数的文件在原项目中的路径、函数内容的局部敏感哈希值和定义了该函数的版本号四个信息。

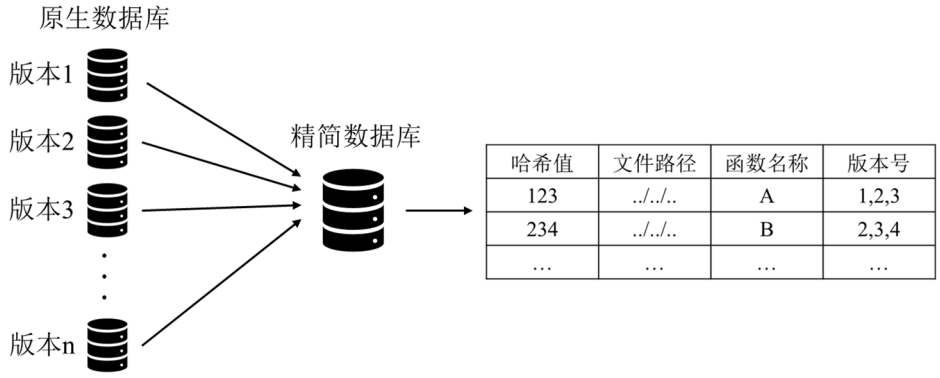


图 2-10　消除冗余获得精简数据库示意图

函数版本号信息是精简数据库和原生数据库最大的差异，也是消除冗余操作的体现。在遍历原生数据库时，如果函数与精简数据库中某个函数相似或相同，则只需要该函数的在版本号一栏添加当前版本号；如果是新函数，则将该函数的函数名字、定义函数的文件在原项目中的路径、函数内容的局部敏感哈希值和定义该函数的版本号四个信息存储到新条目中。生成目标开源软件精简数据库的算法如表 2-4 所示，该算法本质上就是同时遍历原生数据库和正在构建中的精简数据库，利用差异长度算法计算函数局部敏感哈希值之间的相似度，再利用相似度结果不断完善精简数据库的过程。

生成精简数据库算法基于 python 语言，调用了 tlsh 库和 os 库，os 库是 python 中进行文件操作的标准库。由于原生数据库数量多，导致项目结构呈现出复杂的树状结构，因此需要引入 os 库来遍历和读取指定文件夹，并将结果保存到指定位置。另外，本算法使用了 tlsh 库提供的 diffxlen 函数来计算局部敏感哈希值之间的相似

度，与编辑距离等方法相比，具有较快的速度。

表 2-4　生成目标开源软件的精简数据库算法

目标：生成目标开源软件的精简数据库
输入：目标软件的所有原生数据库
输出：目标软件的精简数据库
主函数 main： 新建精简数据库，初始化为空 for（遍历存储目标开源软件的所有原生数据库的文件夹）： 　　读取原生数据库，记录版本号 　　if（精简数据库为空）： 　　　　for（遍历原生数据库的所有条目）： 　　　　　　读取条目，解析得到函数名称，文件路径，局部敏感哈希值 3 部分内容 　　　　　　新条目包含函数名称，文件路径，局部敏感哈希值，版本号 4 部分内容 　　　　　　将新条目写入精简数据库 　　else： 　　　　for（遍历原生数据库的所有条目）： 　　　　　　读取条目 　　　　　　解析得到函数名称，文件路径，局部敏感哈希值 3 部分内容 　　　　　　for（遍历精简数据库的所有条目）： 　　　　　　　　利用差异长度算法计算原生数据库与精简数据库对应条目局部敏感哈希值的相似度 　　　　　　　　将相似度与阈值进行对比 　　　　　　　　if（相似度小于等于阈值）： 　　　　　　　　　　if（原生数据库和精简数据库中对比条目的函数名称相同）： 　　　　　　　　　　　　将当前版本号添加到条目的版本号内容中 　　　　　　　　　　if（原生数据库和精简数据库中对比条目的函数名称不同）： 　　　　　　　　　　　　在该条目下添加函数名称，文件路径信息 　　　　　　　　　　if（差异度大于阈值）： 　　　　　　　　　　　　新条目包含函数名称，文件路径，局部敏感哈希值，版本号 4 部分内容 　　　　　　　　　　　　将新条目写入精简数据库 　　　　　　保存精简数据库，命名为（开源软件名称+.sig）

2.4　本章小结

本章设计了一个自动化脚本，实现了批量下载目标开源软件各个版本的源代码的功能。在得到源代码的基础上，利用 ctags 工具对源代码中的 C 语言、C++文

件进行静态分析，得到文件中所定义的函数名字和该函数的具体内容，再对源代码进行了文本预处理，去除了不需要的文本信息，如注释、制表符、换行符等，屏蔽了因代码书写者习惯不同而引起的相同函数的文本差异。然后利用局部敏感哈希函数处理预处理后的函数内容，得到函数的局部敏感哈希值，组合得到开源软件各个版本的原生数据库。在此基础上对某种开源软件的所有原生数据库进行消除冗余操作，得到最终的精简数据库，精简数据库与开源软件呈一一对应关系，且能被后续阶段直接使用。

本章算法的具体实现基于 python 语言，主要使用了 python 中的提供局部敏感哈希功能的 tlsh 库、调用外部工具的 subprocess 库、提供正则表达式功能 re 库和用于文件操作的 os 库，编写了数百行 python 代码实现了数据库的构建。

对构建的精简数据库的内容进行查看分析，发现精简数据库能够将不同版本中定义的相同或相似的函数映射到数据库的同一位置，证明了本章构建精简数据库算法的有效性。

第 3 章　函数特征的提取

3.1　引言

精简数据库中包含了开源软件中定义函数的名称、哈希值等信息，但是并不包含函数对的信息。LibDetective 框架引入了区域匹配技术，该框架需要对函数对进行匹配，得出函数重用关系和开源软件重用关系。因此，从精简数据库和源代码中提取出函数的特征，是文章的主要任务。

函数的特征可以分为文本特征和结构特征，如图 3-1 所示，文本特征是函数的表层特征，主要作用是快速缩小后续阶段的函数匹配范围；结构特征是函数的深层特征，也是区域相似度比较的基础。本章通过提取并比较函数对之间文本特征的相

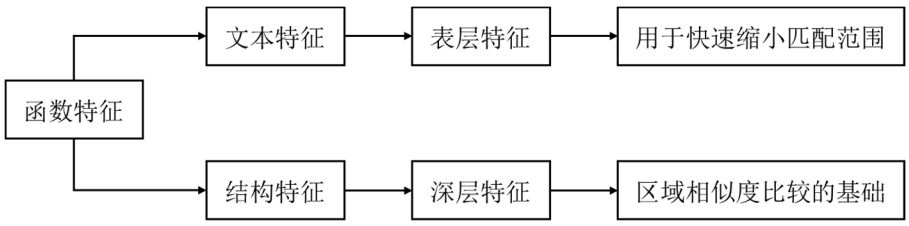


图 3-1　函数特征的分类和作用

似度，得到相似函数对。在生成相似函数对的基础上，又通过构建函数调用关系图来引入区域匹配技术，提取函数的结构特征。区域匹配技术是一种新兴技术，也是本文研究和实现的重点。提取函数中的特征性区域，利用合适的算法计算区域之间的区域相似度并与阈值相比较，即可判断函数的相似度，从而判断出重用关系。有很多种图能够代表函数的结构，例如函数调用关系图、控制流图、控制流依赖图、数据依赖图等，它们都能表示函数的特征。由于开源软件源代码重用时，有可能出现重命名变量、重命名函数、控制语句修改等现象，而调用函数基本不会被修改，因此选择函数调用关系图来表示函数的区域、代表函数的特征是合理的选择。本章探索和实现了生成函数调用关系图的几种方式，并对这几种方式的实现效果进行了比较，选择出更优秀的算法。

本章的工作流程如图 3-2 所示，函数文本特征的提取与函数结构的特征的提取两个步骤是独立的，并不互相作为前提，因此二者可以同时进行以加快工作效率。函数结构的特征的提取可以分为提取文本形式的文件的函数调用关系图和提取有向图形式的函数的函数调用关系图两个步骤，其中获得文件的函数调用关系图是提取函数的函数调用关系的前提，二者存在先后顺序，不能同时进行。

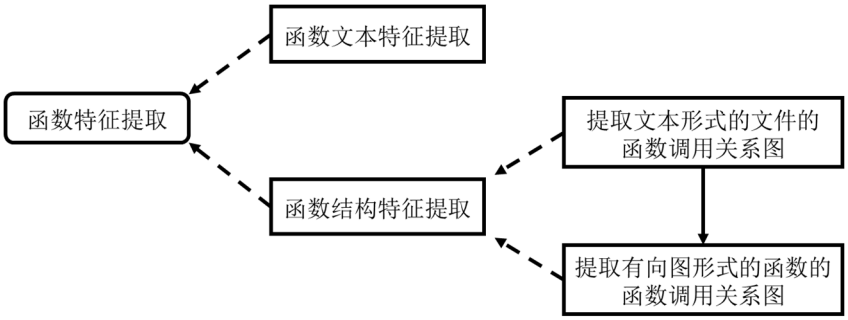


图 3-2　函数特征提取阶段工作流程

3.2　函数文本特征的提取

相似函数对的提取过程实质上是目标软件精简数据库与开源软件精简数据库之间的直接匹配过程，函数文本特征提取的工作流程如图 3-3 所示。从目标软件精简数据库中提取一个函数，称为目标函数，设定一个空列表，使其长度为 k（本文中 k 取 50）。遍历开源软件精简数据库中的所有函数，称为候选函数，利用差异长

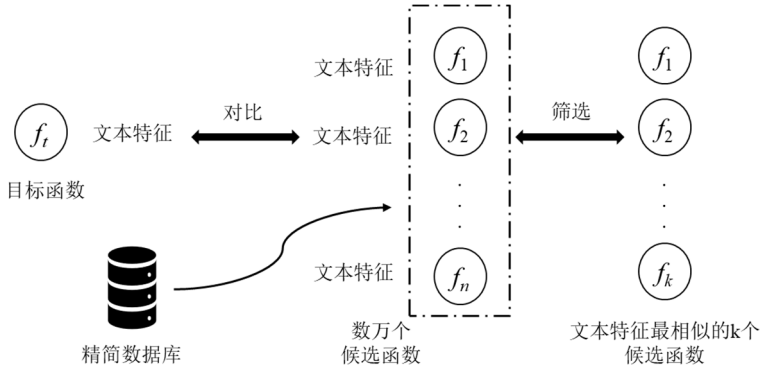


图 3-3　函数文本特征提取工作流程

度算法计算出目标函数与候选函数之间的文本特征的差异值，如果列表还有空位，直接将差异值和候选函数加入到列表中；如果列表已经有了 50 个元素，则找到列表中差异值最大的元素，用当前的差异值和候选函数替换该元素。遍历完成后，对列表进行升序排列，即可得到与目标函数相似度最高的 50 个函数。对整个目标软件精简数据库进行遍历，即可得到目标软件中所有函数的相似函数对。提取相似函数对的算法如表 3-1 所示。

表 3-1　提取相似函数对算法

目标：得到目标软件精简数据库中与每个函数最相似的 50 个候选函数
输入：目标软件的精简数据库和开源软件的精简数据库
输出：目标软件中与每个函数最相似的 50 个候选函数
主函数 main：
for（当目标软件精简数据库中还有函数未进行匹配）

表 3-1（续表）

目标：得到目标软件精简数据库中与每个函数最相似的 50 个候选函数
输入：目标软件的精简数据库和开源软件的精简数据库
输出：目标软件中与每个函数最相似的 50 个候选函数
读取目标函数名称和局部敏感哈希值 创建一个长度为 50 的空列表 for（当开源软件精简数据库中还有函数未进行匹配）： 读取候选函数名称和局部敏感哈希值 利用差异长度算法计算差异值 if（列表有空元素）： 直接把目标函数的名称和差异值写入列表 else： if（差异值比列表中差异值的最大值小）： 替换差异值最大的元素的函数名称和差异值 if（列表中非空元素长度等于 50）： 写入结果 else： 将空元素用 NULL 替代 写入结果 将所有结果合并，得到整个目标软件与开源软件库之间相似的函数对

提取函数文本特征算法基于 python 语言，主要使用了 tlsh 库、os 库和 json 库，json 库是 python 的标准库，用于处理 JSON 数据，它提供了一组用于将 python 数据结构存储为 JSON 格式的字符串和将 JSON 格式的字符串读取为 python 数据结构的函数，这两个过程称为序列化和反序列化。在文本特征对比的过程中需要处理和遍历大量文件，因此需要利用 os 库。tlsh 库也得到了使用，目标函数文本特征与候选函数文本特征相似度的计算由 tlsh 库提供的 diffxlen 函数实现。最终生成的相似函数为键值对的形式，键的内容为匹配的函数名称以及定义该函数的开源软件名称，值的内容为函数的相似度得分，json 库能够在存储键值对的同时，保留键值对之间的对应关系，同时读取 json 文件后能直接得到字典形式的键值对，而不必进行格式转换，提高运行效率。因此文本特征最相似的 k 个候选函数以 JSON 格式保存，json 库提供了 dumps 函数保存信息到指定文件，同时提供了 loads 函数读取信息，具体的保存过程如图 3-4 所示。

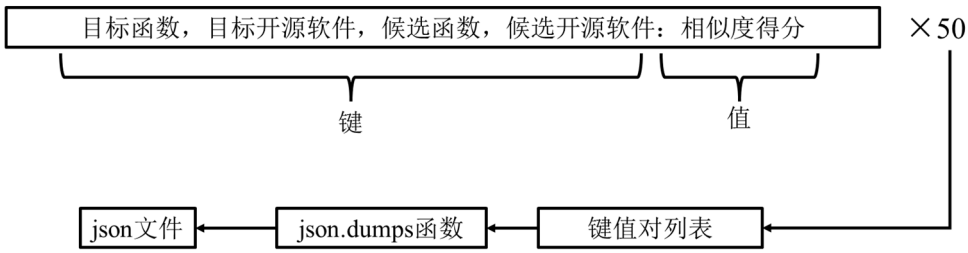


图 3-4　json 库保存键值对列表示意图

3.3　函数结构特征的提取

现有的对 C 语言文件进行静态分析的工具很多，有的可以直接生成文件的函数调用关系图，如 cflow 工具；有的只能生成抽象语法树，需要另外设计算法，分析抽象语法树，得到函数调用关系图，如 tree-sitter 库。本文同时测试了两种方法，并且对比了二者的准确度和运行速度，并选择效率更高的调用静态分析工具生成函数调用关系图的方法作为 LibDetective 框架提取函数结构特征的算法。

3.3.1　深度优先算法生成函数调用关系图

本文测试了利用 tree-sitter 库生成函数调用关系图的效果与速度。tree-sitter 库能够解析源代码，生成对应文件的抽象语法树，在此基础上，可以利用深度优先算法遍历抽象语法树，生成文件的函数调用关系图。获得文件的函数调用关系图后，需要对其进行处理，得到函数的函数调用关系图。

tree-sitter 库解析得到的文件抽象语法树以类似结构体的形式存储着节点的信息，结构体中的属性包括节点类型、节点名称、节点内容、节点内容起始位置和结束位置等信息，父节点的节点内容是该父节点的直接孩子节点的节点内容的组合。利用 tree-sitter 库解析样本文件，分析得到的抽象语法树，可发现定义函数节点的节点类型总是 function_definition，因此只需要遍历抽象语法树并寻找所有节点类型为 function_definition 的节点，即可找到抽象语法树中定义函数的区域，继续对该区域进行深度优先遍历，寻找该函数所调用的其他函数的名称，即可生成函数调用关系图。通过查看 tree-sitter 库源代码中所定义节点类型，发现 tree-sitter 库定义了数十种可能存在函数调用的节点类型，使用 tree-sitter 库生成函数调用关系图方法的局限性开始显现，定义函数的节点包含大量无关调用的子节点，这些子节点的内容可能是符号、变量或者语句等，对应的节点类型种类繁多，此时需要判断节点是否能够产生函数调用，由于有数十种可能存在函数调用的节点类型，如果使用程序的条件分支来处理此任务，可能会连续出现数十条 if 语句。同时，由于需要函数调用关系图是有向图，需要考虑节点先后顺序等逻辑关系，深度优先算法成了合

适的选择。对于文本内容丰富的函数，其抽象语法树会非常复杂，深度甚至能超过10，此时深度优先算法是非常缓慢。本文通过观察与实验，总结出了最常用的调用函数的节点类型，能够将条件分支的数量减小到个位数，但这样的速度提升是以精度的牺牲为前提的，此时获得的函数调用关系图并不完全准确，同时此函数调用关系图的深度也不超过 3。

函数的函数调用关系图和文件的函数调用关系图之间的关系如图 3-5 所示，其中 R 为根节点，表示文件名，其余的大写字母代表函数的名字。函数的函数调用关系图是文件的函数调用关系图的子图，前者的根节点是后者的直接的孩子节点。将一个文件下面所有函数的函数调用关系图组成森林，再添加一个根节点，添加从根节点指向函数节点的边，就形成了文件的函数调用关系图。

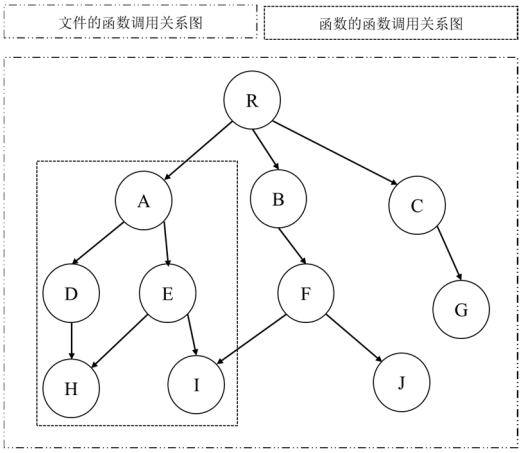


图 3-5　文件的函数调用关系图和函数的函数调用关系图的关系

利用深度优先生成函数调用关系图的算法基于 python 语言，调用了 tree-sitter 库、os 库和 networkx 库。tree-sitter 库是一个用于语法分析的现代化库，它能够高效地、准确地解析源代码并构建抽象语法树（AST），基于抽象语法树可以生成函数调用关系图。利用 tree-sitter 库生成源代码文件的抽象语法树的流程如图 3-6 所示，tree-sitter 库提供了各种语言的语法分析规则，编译语法规则即可得到语言的语法文件，将语法文件输入 tree-sitter 库的 parser 类，即可得到解析器实例，解析器实例提供了 parse 函数，能够解析指定语言的源代码并生成该文件的抽象语法树，抽象语法树是许多节点实例的组合。

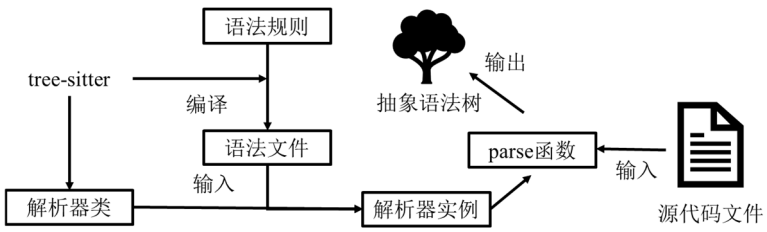


图 3-6　tree-sitter 库解析生成源代码文件抽象语法树的流程

如果需要遍历抽象语法树，则通过抽象语法树类的 root 属性获取该树的根节点，根节点也是节点类的一个实例。tree-sitter 库中定义的节点类包含 go_to_first_child、go_to_next_child 等用于遍历抽象语法树的函数，使用这些函数就能进行遍历操作。利用深度优先算法遍历语法分析树的示意图如图 3-7 所示，创建一个首先指向根节点的指针，从根节点开始，指针不断移动到当前节点的第一个子节点，如果当前节点没有子节点，则指针移动到该节点的兄弟节点；如果该节点

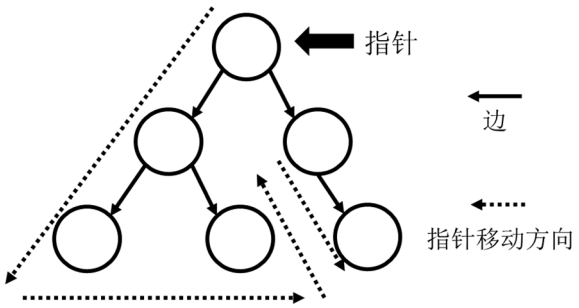


图 3-7　深度优先算法示例图

既没有兄弟节点也没有子节点，则指针回到父节点。父节点、子节点与兄弟节点的关系如图 3-8 所示，A 是 B 和 C 的父节点，B 和 C 是 A 的子节点，B 和 C 互为兄弟节点。

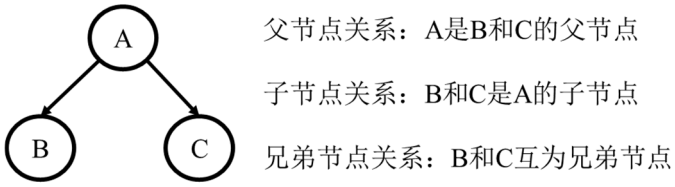


图 3-8　父节点、子节点与兄弟节点的关系案例

networkx 是一种用于创建和操作复杂网络结构的 python 库，它提供了丰富的数据结构和算法，方便用户处理网络结构。函数调用关系图是一种有向图，networkx 库具有支持各种复杂图数据结构的功能，包括有向图。networkx 库中定义的 Digraph 类被广泛地用于表示和处理有向图，该类能保存图节点和边的信息，并且利用图的信息快速地计算节点的深度、节点的度等信息，提高后续区域相似度计算阶段的工作效率。Digraph 类中图的节点信息与边的信息能被保存为 networkx 库专有的存储图的文件格式，后缀名为“.gml”，networkx 库直接提供了 write_gml 方法保存函数为该文件类型，读取时则使用 read_gml 方法直接读取图结构，read_gml 方法返回一个图的实例。使用 networkx 库提供的读写文件函数省去了格式转换的时间开销，直接存储和读取图实例，提高了工作效率。创建新的有向图实例后，可以利用 networkx 库定义的 add_node 方法向实例中添加节点，同时利用 add_edge 方法添加从源节点指向目标节点的边。生成函数调用关系图的算法所使用的 networkx 库中

的函数类型和函数功能如图 3-9 所示，本文主要使用了读写图实例的函数和向图中添加节点和边的函数。

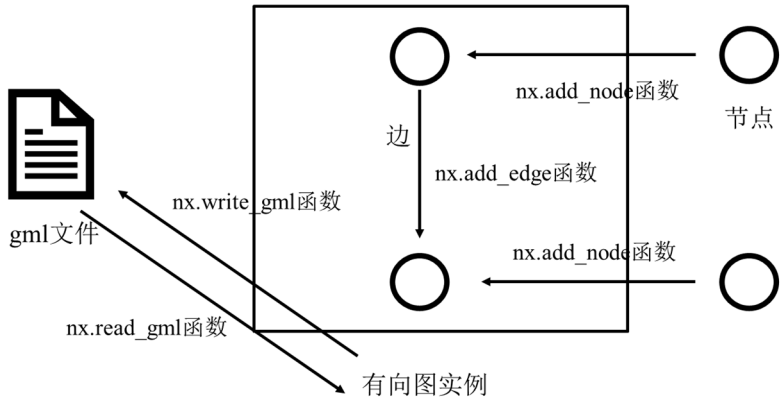


图 3-9　生成函数调用关系图算法使用到的 networkx 库中函数功能示意图

3.3.2　调用静态分析工具生成函数调用关系图

本文使用 cflow 工具生成文件的函数调用关系图，通过解析 cflow 工具的输出结果得到每个函数的有向图形式的函数调用关系图，流程如图 3-10 所示。

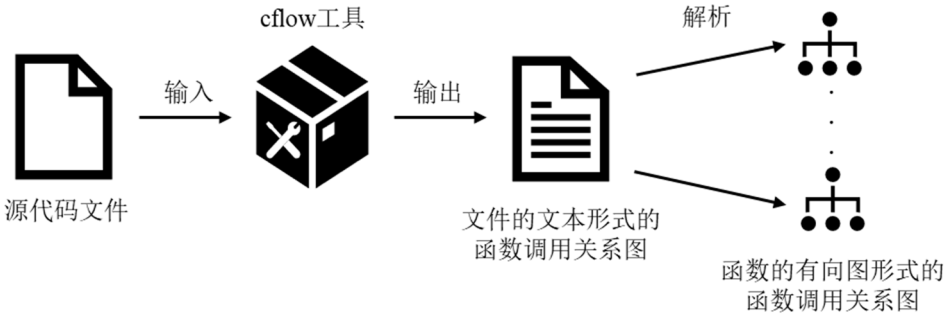


图 3-10　调用工具生成函数调用关系图工作流程

目前主流的能够生成函数调用关系图的静态分析工具如表 3-2 所示，鉴于本文只需要利用静态分析工具生成函数调用关系图，对功能丰富性要求不高的同时需要较快的解析速度，因此 cflow 工具是主流分析工具中较好的选择。cflow 是一款

表 3-2　主流的能够生成 C 语言函数调用关系图的工具对比

	cflow	Doxygen	Graphviz
规模	小	中	小
解析结果	文本格式	UML 格式	DOT 格式
速度	快	慢	慢
功能丰富性	弱	强	强

静态分析的工具，可以通过命令行调用 cflow 解析指定的文件，获得指定的解析结果。在此基础上，本文构造了一个自动化脚本，调用 cflow 工具，自动的、批量的

生成文本形式的文件的函数调用关系图。cflow 解析得到的函数调用关系图有可能存在一些报错信息，同时，在开源软件项目中有很多配置文件或没有定义函数的文件，这些情况需要进行特殊处理。在文本形式的函数调用关系图基础上，对文本文件进行遍历，保存为有向图形式，即可得到最终所需要的函数的函数调用关系图。

cflow 工具解析源代码得到的文件的文本形式的函数调用关系图保存形式为文本形式，该文件的每行由字符串和空格符组成，空格符在字符串之前。字符串代表函数名称，空格符的个数体现该函数所在的深度。在文件的函数调用关系图中，第一行是文件的名字，空格符的数量为 0，代表深度为 0；定义函数的行的空格符的数量为 4，代表深度为 1；其他行的节点与深度的关系如式（3-1）所示，*NS* 代表空格符的数量，深度与空格符的数量呈线性关系。

$$depth = \frac{NS}{4} \tag{3-1}$$

得到文件的文本形式的函数调用关系图后，遍历该文本文件，依次读取每一行的信息，获得空格符的数量和函数名称，通过式（3-1）计算节点深度。由于函数调用关系图是有向图，深度相同的节点不一定属于同一个父节点，因此需要利用一个列表来存储当前节点的父节点。初始时刻利用 networkx 库新建有向图，在不断遍历文本的过程向有向图中添加对应的节点和相应的边，当再次遇到深度为 1 的节点时，说明上一个函数的函数调用关系图的内容已经全部添加完成，保存该有向图，再次创建一个新有向图，重新初始化父节点列表，继续遍历文本文件，当文件遍历完毕时，保存有向图，完成对该文件中所有的函数的有向图形式的函数调用关系图的构建任务，文本形式的函数调用关系图转换为有向图形式的函数调用关系图的原理如图 3-11 所示。

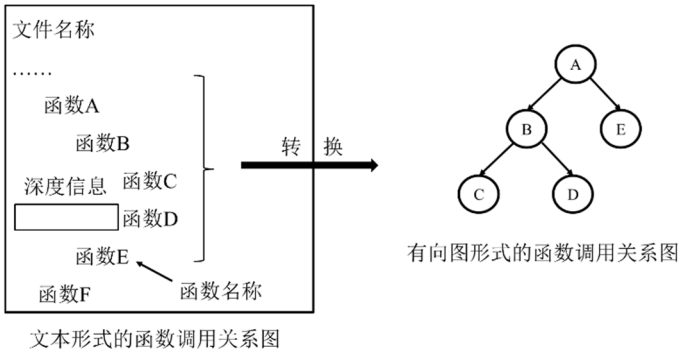


图 3-11　文本形式的函数调用关系图转换为有向图形式的函数调用关系图原理

文本形式的函数调用关系图和有向图形式的函数调用关系图的区别如表 3-3 所示，文本形式的函数调用关系图文件后缀名为.txt，以文本形式存储图的结构并不能直接被后续操作使用。因此需要解析文本，提取文本中存储的图结构并保存为

能直接使用的有向图格式，后缀名为.gml。得到文本形式的函数调用关系图算法的流程如

表 3-4 所示，得到有向图形式的函数调用关系图算法的流程如表 3-5 所示。

表 3-3　文本形式的函数调用关系图和有向图形式的函数调用关系图的区别

	文本形式的 函数调用关系图	有向图形式的 函数调用关系图
文件后缀名	.txt	.gml
能否直接使用	不能	能

表 3-4　获得文本形式的函数调用关系图

目的：调用 cflow 工具，获得文本形式的函数调用关系图
输入：整个开源软件的源代码
输出：所有 C 语言文件的文本形式的函数调用关系图
主函数 main： for（遍历整个项目）： if（文件的后缀名为.c）： 　　构造解析指令 　　try： 　　　　调用 cflow 工具，同时输入指令 　　　　生成文本形式的文件的函数调用关系图，并保存为指定路径 　　catch： 　　　　文件名称，　异常原因

表 3-5　获得有向图形式的函数调用关系图

算法：处理文本形式的函数调用关系图，生成有向图形式的函数的函数调用关系图
输入：所有文本形式的文件的文件调用关系图
输出：有向图形式的函数的函数调用关系图
主函数 main： for（遍历产生的所有文本形式的函数调用关系图）： 　　try： 　　　　获取文本形式的函数调用关系图 　　　　解析 　　catch： 　　　　文件名称，　异常原因 解析（输入：文本形式的文件的函数调用关系图　输出：有向图形式的函数的函数调用关系图）： try：

表 3-5（续表）

算法：处理文本形式的函数调用关系图，生成有向图形式的函数的函数调用关系图
输入：所有文本形式的文件的文件调用关系图
输出：有向图形式的函数的函数调用关系图

```
创建父节点列表，初始化为空
for（遍历文本形式的文件的函数调用关系图）：
    if（此节点深度为 0）：
        if（父节点列表为空）：
            父节点列表在索引为 0 的位置添加该节点
            创建一个新的有向图
            向有向图中添加该节点
        else：
            保存有向图
            父节点列表在索引为 0 的位置替换为该节点
            创建一个新的有向图
            向有向图中添加该节点
    else：
        计算该节点深度，向有向图中添加该节点
        向有向图中添加从父节点列表索引为深度-1 的节点指向该节点的边
        if（父节点列表的长度等于深度）：
            向父节点列表的末尾添加该节点
```

调用 cflow 工具生成函数调用关系图的算法基于 python 语言，调用了 subprocess 库、os 库和 networkx 库。调用 cflow 工具解析源代码生成函数调用关系图的指令由 cflow 可执行文件的路径和待解析源代码文件路径组成，由于 cflow 软件的安装路径在运行过程中是不变的，因此只需要不断利用待解析源代码的路径构造解析指令，利用 subprocess 库的 run 函数调用 cflow 工具，同时将指令输入该工具，即可自动化地生成大量文件的函数调用关系图。算法的具体工作流程如图 3-12 所示，程序脚本对文件夹中的所有源代码文件进行遍历，将源代码文件的路径与固定前缀进行组合生成指令，同时利用指令调用外部工具解析源代码文件，得到函数调用关系图。当文件夹中的所有文件都被解析时，程序停止运行，解析得到的文本形式的函数调用关系图被存储到指定位置，如果存在解析异常的文件，文件路径也被记录到专门的文件中。

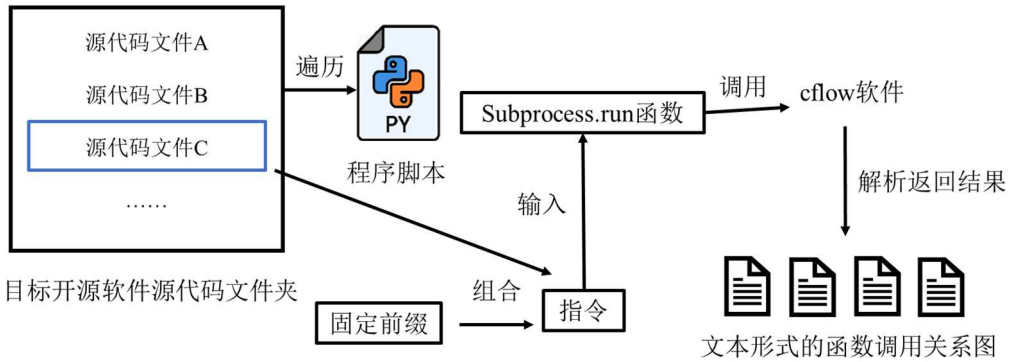


图 3-12　构造指令自动调用 cflow 软件生成文本形式的函数调用关系图算法流程图

3.3.3　方法对比

直接调用工具生成函数调用关系图的方法与遍历语法分析树生成函数调用关系图的方法性能对比如表 3-6 所示，调用工具生成函数调用关系图的方法在速度和准确度上都有优势。由于节点类型的复杂性，用深度优先算法生成准确的函数调用关系图需要利用数十个条件分支，导致效率低下。通过减少条件分支能提高深度优先算法的检测效率，但这样的效率改善是以牺牲精度为前提的。尽管采取了措施提高检测速度，深度优先算法的效率与调用工具算法相比仍然很低，根本原因对树状结构进行遍历的过程本质上是一种递归，在解析比较复杂的文件时，文件的抽象语法树会非常复杂，深度能超过 10，这导致遍历语法树时，递归的深度会超过 python 默认最大次数 997 次，导致解析失败。通过修改默认最大递归次数能够改善这一问题，但如果数据量过于庞大，电脑内存很可能被过度占用导致电脑宕机，程序异常停止。因此，考虑到本工具应该有处理大量数据的能力，调用 cflow 工具生成函数调用关系图是更好的方法。

表 3-6　生成函数调用关系图的两种方法性能对比

	调用工具算法	深度优先算法
速度	快	慢
准确度	高	低

3.4　本章小结

本章对目标软件和开源软件的精简数据库进行遍历，提取了函数的文本特征，同时利用差异长度算法计算函数相似度，得到了与每个目标函数相似度最高的 50 个候选函数，生成了相似函数对，过滤了大量无关函数，极大的减少了后续函数匹配工作的工作量。利用差异长度算法计算函数对局部敏感哈希值的相似度具有较快的速度，而后续区域相似度的比较阶段速度相对较慢，本文通过多进行简单操作

来减少后续复杂操作进行的次数，同时这样的做法并不会影响检测效果，在保证检测准确度的情况下进一步提高算法的工作效率。该算法的实现基于 python 语言，利用了第 2 章中构建的精简数据库，引入了 tlsh 库，利用数百行 python 代码实现了函数文本特征的提取任务

同时，本章分析了目前主流的生成函数调用图的工具的优缺点，并根据本文的任务场景选择了合适的工具，设计了有效的算法生成函数的函数调用关系图，提取了函数的结构特征。通过对比深度优先算法和调用工具法的运行时间和结果准确度，最终选择调用工具生成函数调用关系图的方法，该算法在速度和准确度上都具有显著的优势。该算法的实现基于 python 语言，调用了免费的开源软件 cflow，引入了 python 中的 subprocess 模块和 networkx 模块。本章编写了数百行 python 代码，实现了函数结构特征的提取任务。

第 4 章　区域相似度的比较

4.1　前言

图嵌入技术是将图映射成向量的技术。要比较图的相似度，首先就需要将图映射到相同维度的向量空间。目前有很多种方法可以进行图相似度的比较，例如图编辑距离、图核函数[54]、最大子图[55]、图神经网络[56]等，图神经网络包括了超过 33 种[56]著名的网络。本文需要计算函数的函数调用关系图之间的相似度，遗憾的是，目前虽然存在用于函数调用关系图的图神经网络，但是现有的图神经网络需要构造大量的训练集进行训练，最终的预测能力依赖训练集，容易造成过拟合，且结果不具有可解释性。另外，现存的非图神经网络的算法，如图编辑距离、最大子图等，在面对规模较大的图时，计算速度相当缓慢。本文意在探索一种轻量级的、具有可解释性的图嵌入方法，不需要繁琐的训练过程，能够快速、准确的对大规模函数调用图进行相似度匹配。因此，本文提出并使用了基于节点的度的一种新的图嵌入方法，实现了快速的、可解释性的进行图相似度匹配。

本章设计的区域相似度比较算法包括生成特征向量、特征向量归一化和计算余弦相似度 3 个步骤，匹配流程如图 4-1 所示。特征向量是函数调用关系图结构在另一种空间中的存在形式，函数调用关系图自身的性质决定了特征向量的尺寸、方向等。由于函数调用关系图节点数量的差异，生成的特征向量之间的内积的取值范围并不确定，因此需要对特征向量进行归一化，使得特征向量之间内积的取值范围在-1 到 1 之间。归一化后的特征向量之间的余弦相似度代表了函数调用关系图的相似度，余弦相似度是进行区域相似度比较的判据。

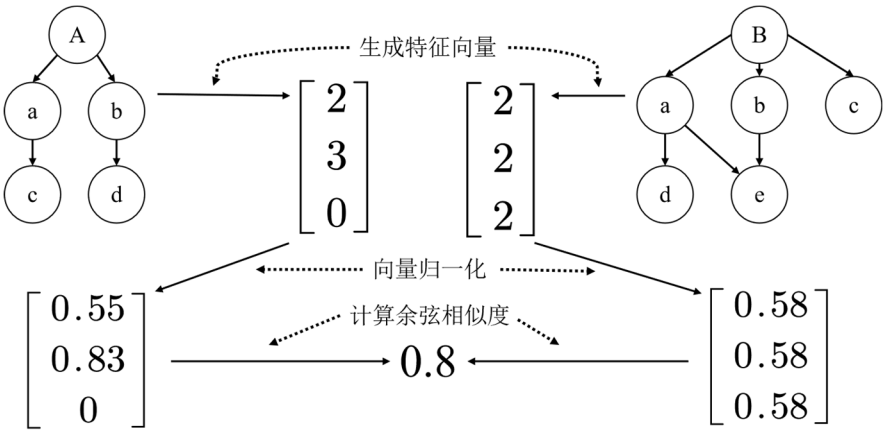


图 4-1　计算函数关系图相似度的流程

4.2 基于节点度的图嵌入

图是一种在物理空间中抽象的、很难量化的数据结构，要定量地比较图的相似度，必须提取出图的某种特征，并利用合适的数学手段量化该特征，再基于图之间的特征进行运算，最后利用运算结果来表示图之间的相似度。常用的方法是利用向量来表示图的特征，该向量是图的特征在另外的数学空间中的体现，而将图转化为向量的过程称为图嵌入，图嵌入得到的向量被称为特征向量。图嵌入算法的本质如图 4-2 所示，本质就是将物理空间中的图映射为数学空间中的向量，将物理空间中难以解决的问题迁移到容易解决的数学空间中进行处理。

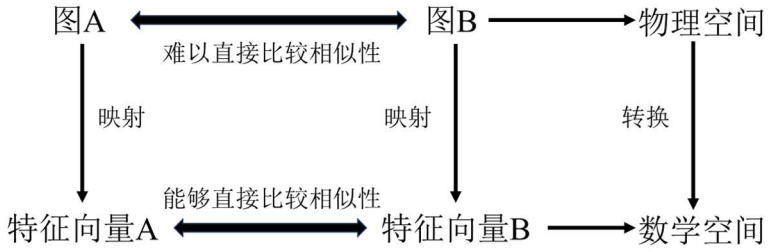


图 4-2 图嵌入算法的本质

本文使用一种基于节点的度的图嵌入方法来生成函数调用关系图的特征向量。图中节点的度，指的是与某个节点相连的边的个数。由于函数调用关系图是有向图，在这种场景下，度又分为出度和入度，出度指的是以该节点为起点的边的个数，对应地，入度指的是以该节点为终点的边的个数，度、出度和入度之间的计算案例如图 4-3 所示。在进行图嵌入算法时，考虑到得到的函数调用关系图存在许多出度或入度很小的节点，将两种度分开考虑，得到的向量维度会翻倍，但是向量上的各个元素会变得更小，计算相似度得到的值有可能会变得很小，导致结果不准确。因此，尽管函数调用关系图是一种有向图，本文还是只利用其节点的度进行图嵌入，不对出度与入度加以区分。

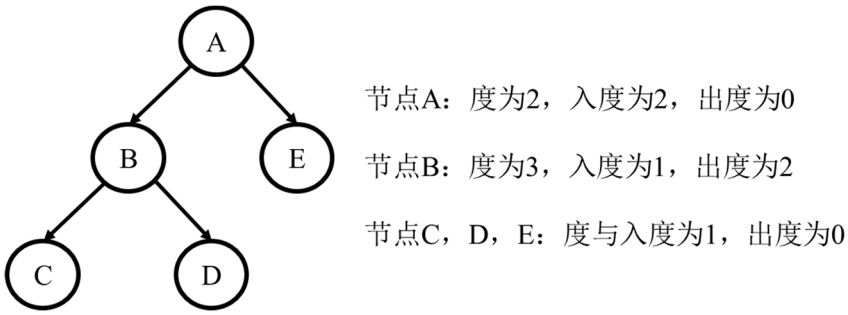


图 4-3 度、入度和出度示意图

本文提出的基于节点度的图嵌入算法包括两个步骤：根据函数调用关系图生成特征向量，然后对特征向量进行归一化。

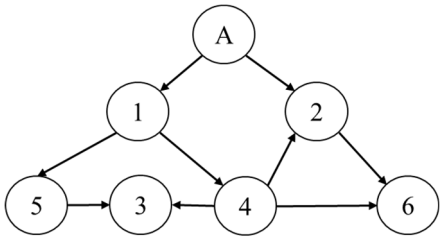
4.2.1 特征向量的生成

函数调用关系图的特征向量指的是在其他数学空间中表示该图某种特征的向量，本文设计的生成特征向量的算法如表 4-1 所示。读入函数调用关系图后，首先计算函数调用关系图中各个节点的度，然后统计各个图中，每一种数量的度的节点的个数，例如，有 3 个节点度为 1，有 2 个节点度为 2 等。最后选取两幅图中存在的度的最大值，作为要创建的特征向量的维度，例如，图 A 中节点的最大度数为 5，图 B 中节点的最大度数为 7，则特征向量的维度为 7，在创建向量时，图 A 的特征向量被扩充到 7 位，高位的空位用 0 补充。

表 4-1 生成特征向量算法

算法：利用函数调用关系图，生成特征向量
输入：一对函数调用关系图
输出：一对特征向量
主函数 main：
读取函数调用关系图 A
读取函数调用关系图 B
计算图 A 中各个节点的度，统计图 A 中每种数量的度的节点个数
计算图 B 中各个节点的度，统计图 B 中每种数量的度的节点个数
特征向量维度 = 最大值（图 A 中最大的度，图 B 中最大的度）
创建向量
return：
图 A 和图 B 的特征向量

可以看到，这里的特征向量尺寸并不是完全不变的，特征向量的尺寸取决于需要对比的两个函数调用关系图。如果将向量的维度设置为定值，维度设定值过大，可能会引起向量稀疏、空间浪费等问题；维度设定值过小，可能会产生度数截断问题，导致度数超过一定值的节点的度在向量中会被舍弃或者合并，造成信息的丢失和错误，使得准确度降低。如图 4-4 所示，如果设置向量的维度为定值 3，则会导



理论上的特征向量：$[0\ 4\ 2\ 1]^T$
设置向量维度为定值3
得到的特征向量：$[0\ 4\ 2]^T$

图 4-4 度数截断现象的案例

致度为 4 的节点的信息被舍弃，使特征向量不准确，从而影响检测精度。鉴于不同函数的函数调用关系图复杂性可能存在巨大差异，很难设定一个固定值同时满足节约空间和保证精度两个要求，因此，动态设置特征向量尺寸是更好的选择。

4.2.2 特征向量归一化

特征向量经过归一化后形成基向量，基向量能屏蔽函数调用关系图节点数量的差异，为后续处理带来便利。归一化的原理如式（4-1）所示，生成的特征向量需要除以其本身的 2-范数，得到模为 1 的基向量。这个相除的过程称为归一化。

$$\boldsymbol{v}=\frac{\boldsymbol{V}}{\|\boldsymbol{V}\|} \tag{4-1}$$

函数的函数调用关系图节点数量一般不为 1，因此直接生成的特征向量的 2-范数一般不为 1。假如不进行特征向量的归一化而直接计算向量之间的内积，得到的结果值域是不可控的，如图 4-5 所示，图中两个图的特征向量的内积为 10，对于不同的图，内积的值会无规律的出现，这样的不确定性为图相似性的比较带来了困难。特征向量归一化将产生的特征向量转换为基向量，基向量之间的内积范围是确定的，引入阈值与基向量之间的内积相对比，即可判断向量之间的相似性。因此，特征向量归一化对基于余弦相似度的图相似度比较奠定了基础。

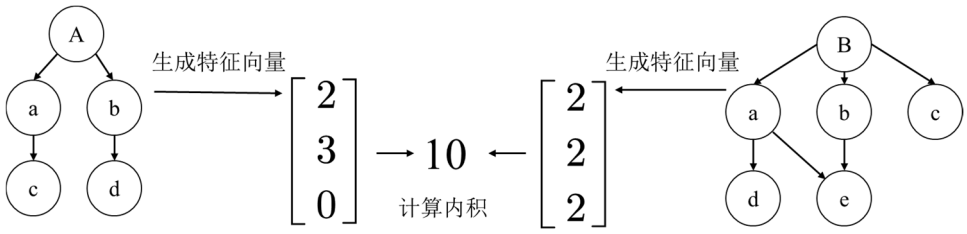


图 4-5　内积值域不可控示意图

4.3 基于余弦相似度的图相似度比较

在本文提出的基于节点的度的图嵌入算法中，向量是成对产生的。产生向量的维度是两张图中度最大的节点的度的值。归一化之前，向量中处于第 n 位数代表该图中度数为 n 的节点的个数；归一化之后，向量中处于第 n 位数代表该图中度数为 n 的节点的个数占总节点数的比例。比较向量之间相似度的方法有很多，余弦相似度是其中很常用的一种。归一化后，一对基向量的余弦相似度得分便是二者的内积，如式（4-2）所示。

$$score=\boldsymbol{u}\cdot\boldsymbol{v} \tag{4-2}$$

余弦相似度的意义是向量空间中一对向量之间夹角的大小，图嵌入操作将图映射到其他空间，向量所指向的方向代表图的特征，越相似的图在该空间中向量的

指向越相似。因此在三维空间中难以比较的图的相似度在其他空间中被量化为了向量之间夹角的大小，越相似的向量对二者的夹角越小。余弦函数具有这样的性质，空间中夹角越接近的向量对二者之间的余弦值越大，满足要求，因此，本文选择余弦相似度作为向量之间相似度的度量标准。

本文提出的基于节点度的图嵌入具有明确的物理意义，向量中的每个元素表示相同度数的节点数量占总节点数量的比例。因此根据 Steck[57]等人的结论，本文将函数调用关系图映射到了一个可解释的向量空间，图嵌入产生的向量空间在空间中的指向由明确的物理性质决定，本文使用余弦相似度作为函数调用关系图之间相似度的度量具有可解释性。

本文使用的图嵌入算法基于 python 语言，调用了 networkx 库和 numpy 库。networkx 库提供的 Digraph 类中定义了 degree 属性，调用该属性即可得到函数调用关心图中每个节点的度，统计每种节点的度出现的次数，并按照顺序排列于列表中，例如，表示节点度为 3 的位置处于列表的第 3 个位置，并记录两个图中节点的度的最大值。在向量归一化和区域相似度比较阶段，使用了 numpy 库提供的 array 函数、norm 函数和 dot 函数。使用 array 函数将列表转换为维度为节点度公共最大值的向量，该向量除以由 norm 函数计算得到的向量的 2-范数即可得到归一化后的基向量，dot 函数计算基向量之间的内积，即可得到图之间的相似度得分。本文计算函数调用关系图之间余弦相似度的具体实现流程如图 4-6 所示。

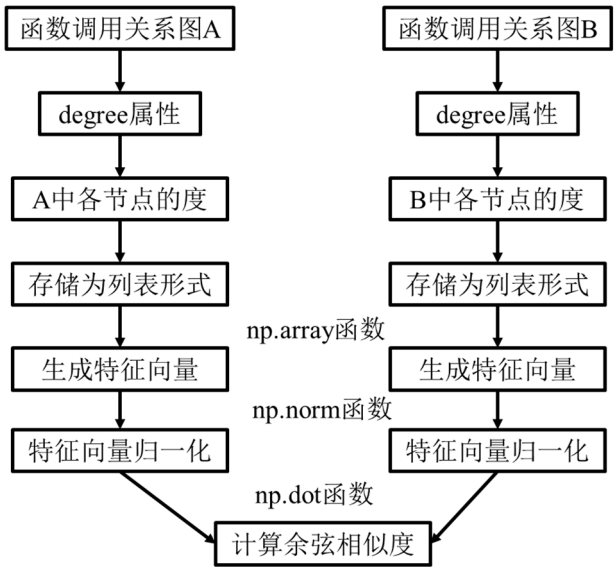


图 4-6　计算函数余弦相似度的流程

4.4　修正系数的产生

经过对结果的观察和研究，作者发现仅靠 4.2 节中的方法，无法处理度的低维

堆积现象。低维堆积现象指的是在生成函数调用关系图特征向量的过程中，部分复杂函数调用关系图存在大量的叶子节点或只有一个孩子节点的节点，导致生成的向量在第 1 个位置或第 2 个位置出现值很大的数，与简单的函数调用关系图计算余弦相似度时，由于一对向量的值集中在低维第 1 或者第 2 个位置，导致原本差异很大的图计算得到的相似度得分很高，使得检测不准确。

例如图 4-7 所示，图 A 仅有 1 个节点，图 B 有 7 个节点，但是二者之间的余弦相似度能够达到 99%，这显然是不正确的，因此需要在 4.2 节的方法基础上添加一个修正系数进行校正，以降低相似度，消除低维堆积现象。在此情况下，最终的相似度得分等于修正系数与余弦相似度得分的乘积，如式（4-3）所示。

$$similarity = \beta \times score \tag{4-3}$$

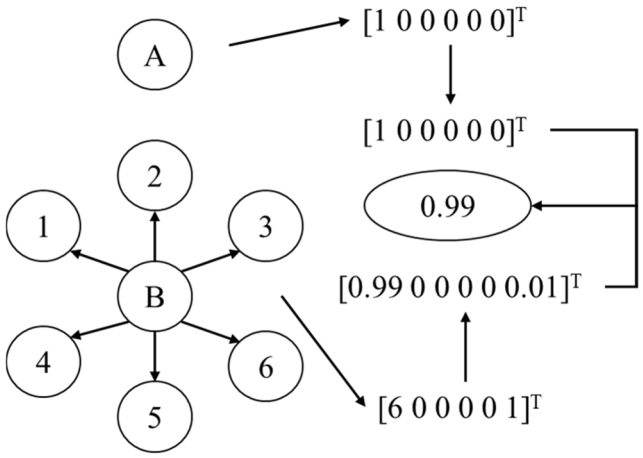


图 4-7　A 和 B 函数调用关系图产生低维堆积现象的案例

如果修正系数是一个定值，检测准确度在总体上并不会提高，因此应该动态的选择修正系数。观察到图 A 和图 B 的节点数量相差很大，可以选择修正系数为：

$$\beta = \frac{2\min(NA, NB)}{NA + NB} \tag{4-4}$$

在式（4-4）中，修正系数为 β，NA 和 NB 分别表示图 A 和图 B 节点的数量。式（4-4）的分子是图 A 的节点数量和图 B 的节点数量之间的最小值的两倍，分母是图 A 的节点数量和图 B 的节点数量。

同时，也可以选择如式（4-5）所对应的修正系数：

$$\beta = \exp(\frac{\min(NA, NB)}{0.5(NA + NB)}) \tag{4-5}$$

式（4-5）使用了自然对数 e，指数的分子是图 A 和图 B 节点数的平均值，分母是图 A 和图 B 节点数的最小值。

引入修正系数以后，选择式（4-4）定义的修正系数，得到的最终的相似度为

25%；选择式（4-5）定义的修正系数，得到的最终的相似度为 2%，均能得到较低的相似度，低维堆积现象便会被过滤，检测准确度得以提高。

可以观察到，当两个图节点数相同时，修正系数为 1；当两个图节点数量相近时，修正系数接近 1；当两个图节点数量相差很大时，修正系数很小。动态产生的修正系数利用两图节点数量的关系产生权重，利用权重调节对图直接进行相似度计算得到的得分，很好的解决了由低维堆积现象引起的相似度得分不准确问题。

4.5 本章小结

本章研究并提出了一种基于节点的度的图嵌入算法，并提出了动态修正因子作为图嵌入算法的补充。本章提出的图嵌入算法基于函数调用关系图的节点的度，基于图中各种度的节点数量生成函数调用关系图的特征向量，其中，特征向量的维度是动态可变的，与需要匹配的两个图有关。同时如图 4-8 所示，本文发现了可能造成结果不准确的两种现象，并提出了解决方案，消除了两种现象造成的干扰，低维堆积现象可以通过引用修正因子来解决；维度截断现象可以通过动态化向量的维度来解决。本文通过提出的基于节点的度的一种图嵌入算法和引入的动态修正因子，很好的完成了区域相似度比较任务。至此，LibDetective 算法设计工作已经完成，后续工作时基于该算法的工具开发和与主流框架的对比实验。

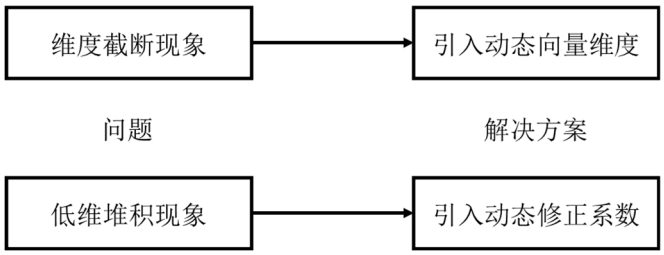


图 4-8　本章发现的问题与对应解决方案

第 5 章　检测工具的开发与性能测试

5.1　前言

LibDetective 算法的实现是本文所做项目的部分工作，设计和构建以该算法为内核的第三方库成分分析工具并在该工具的基础上与主流算法进行对比实验，是本项目的最终目的。本章综合分析软件任务场景，选择合适的软件形式，进行软件的开发，搭建 LibDetective 框架。

目标软件与开源软件之间的重用关系主要有 4 种，可以分为简单重用关系和复杂重用关系。复杂重用关系是影响检测精度的重要因素，检测复杂重用关系是软件组成分析算法的难点。为了测试 LibDetective 框架的性能，对比实验是必不可少的；为了证明 LibDetective 框架能够检测复杂重用关系，侧重于检测复杂重用关系的数据集是必不可少的。在和 LibDetective 应用场景相同的框架中，最具代表性的是 Centris[43]和 TPLite[44]。二者对比如表 5-1 所示，TPLite[44]原理上与 Centris[43]相似，最大的创新点是引入了动态阈值，在匹配粒度上并无创新，因此 TPLite[43]虽然检测效果优于 Centris[43]，但同样面临对复杂重用情况检测失灵的问题。因此本文选取 Centris[43]作为对比，选取了三组侧重于检测不同重用关系的数据集，对 Centris[43]和 LibDetective 同时进行测试，统计二者的检测结果，计算出了准确率、精确率和召回率，验证了 LibDetective 算法能够检测复杂重用关系，证明了 LibDetective 框架的优秀性。

表 5-1　Centris 和 TPLite 对比

	Centris	TPLite
匹配粒度	函数级粒度	函数级粒度
二者的区别	固定阈值	动态阈值
提出年份	2021	2023
能否检测复杂重用关系	否	否

5.2　软件的开发

本文设计的软件组成分析工具工作流程如图 5-1 所示，本文设计并实现了一款基于浏览器/服务器架构的软件，网站就是一种典型的基于浏览器/服务器架构的软件，本文实现的成分分析工具本质上相当于一个网站。选择 Apache 作为服务器软件，Flask 作为 Web 应用框架，完成了前端后端代码的编写。经过测试，该软件能够检查用户输入，如果用户输入满足要求，则正常解析且返回解析结果；如果用户

输入不符合要求，则给出提示和指导。

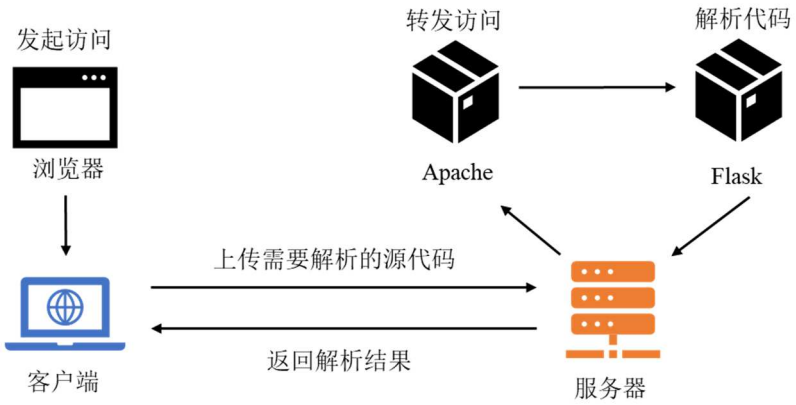


图 5-1　本文设计的软件组成分析工具工作流程图

目前软件市场上流行的软件主要包括客户端/服务器架构和浏览器/服务器架构，这两种形式各有优缺点，二者的工作流程如图 5-2 所示。客户端/服务器架构是一种传统的系统架构模式，其中客户端（Client）和服务器（Server）之间进行通信和交互。客户端负责发送请求并接收服务器的响应，而服务器负责处理这些请求并提供相应的服务或数据。客户端/服务器架构通常包括两个主要组件：客户端应用程序和服务器应用程序。客户端应用程序通常运行在用户的本地设备上，例如个人电脑、移动设备等，而服务器应用程序则运行在远程服务器上。浏览器/服务器架构是一种基于 Web 的系统架构模式，其中用户通过 Web 浏览器（Browser）与服务器进行通信和交互。用户通过浏览器访问 Web 应用程序，而服务器提供 Web 页面和相关的服务。浏览器/服务器架构将应用程序的逻辑部分（服务器端）和用户界面部分（客户端）分离，用户无需安装任何额外的软件，只需通过浏览器即可访问 Web 应用程序。两种架构都取得了广泛的应用，客户端/服务器架构的软件具有更好的性能与安全性，但是其开发难度大，不能跨平台使用，并且由于软件同时存在于客户端和服务器端，软件更新比较繁琐；与此同时，浏览器/服务器形式的软

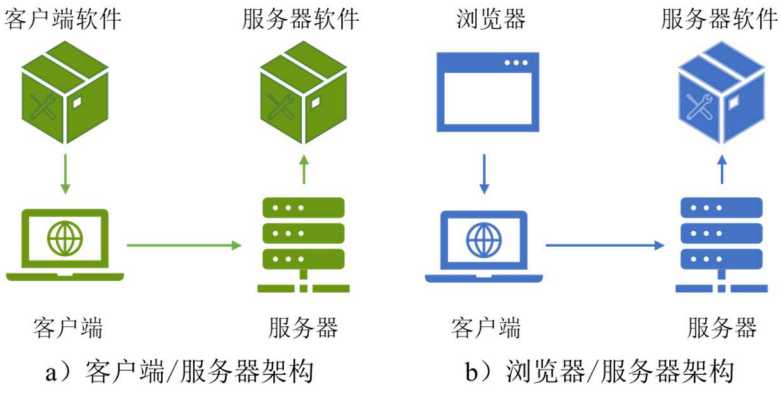


图 5-2　客户端/服务器架构和浏览器/服务器架构对比图

件具有易于维护、可以跨平台使用和易于部署的优点，缺点是安全性和性能受限。本项目预设达到的效果包括：能在不同的操作系统下运行，能够快速部署，能够有很好的可扩展性。因此，本文选择浏览器/服务器架构作为本项目所构建的第三方库检测工具的架构。

目前有许多 Web 服务器软件可供选择，使用最广泛是 Nginx 和 Apache，二者性能上的优缺点如表 5-2 所示。Apache 是最早的服务器软件之一，经过长时间的发展和优化，已经非常成熟稳定，且具有扩展性强、功能丰富的特点。尽管与 Nginx 相比，Apache 在性能上有劣势，但鉴于软件的任务场景相对单一，本文选择 Apache 作为服务器软件。

表 5-2　Apache 和 Ngnix 性能对比

	Apache	Ngnix
可扩展性	强	弱
并发性	弱	强
功能丰富性	强	弱

在选择 Web 服务器软件的基础上，本文选择了合适的 Web 应用框架。主流的 Web 应用框架有 Java 语言的 Spring Boot 框架、PHP 语言的 Laravel 框架和 python 语言的 Django 框架、Flask 框架等。本项目的算法是基于 python 语言实现的，分析工具的任务是接受用户输出和调用 python 脚本以实现算法，并且返回解析结果，因此利用 python 语言的框架成为首选。Flask 框架和 Django 框架的对比如表 5-3 所示，Django 在性能和灵活性上稍弱，但是功能齐全，Flask 则相反。鉴于软件任务场景相对单一，并不需要与用户产生频繁的交互，本文选择轻量化的 Flask 框架作为成分分析工具的 Web 应用框架。

表 5-3　Django 框架和 Flask 框架对比

	Django	Flask
性能优秀性	弱	强
功能丰富性	强	弱
灵活性	弱	强

在选择了服务器软件和应用框架的基础上，本文开始进行前后端代码的编写工作，前后端代码分别有不同的侧重点，具体要求如图 5-3 所示。前端代码需要实现直观的软件界面，让用户容易地操作软件。后端代码要考虑异常处理的情况。由于浏览器/服务器架构的软件需要通过网络与用户进行交互，在编写代码时，应该考虑可能发生的异常情况并进行相应的处理。在外界环境上，由于源代码的上传与解析结果的回传都需要网络，因此处理网络不稳定导致的异常。在用户交互上，需要设置基本的防黑客措施，对用户输入进行检查，防止用户上传病毒或木马等，对

于用户错误的输入，后端进行解析并返回提示；如果用户上传的文件正确，也要考虑可能存在的文件解析错误的情况，分析报错信息，提示用户进行相应的处理。

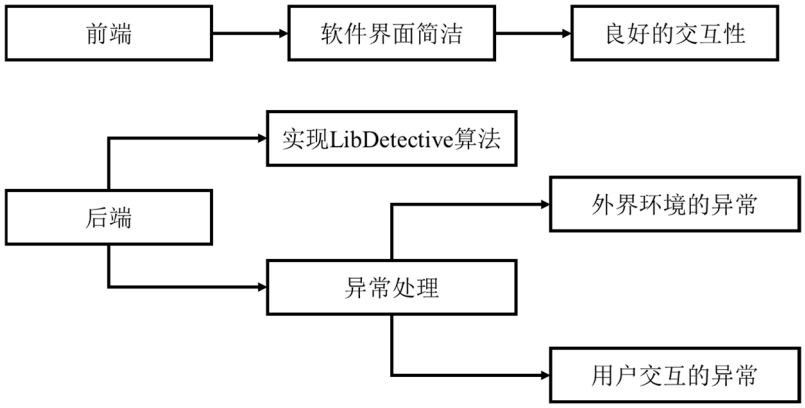


图 5-3　本文设计工具前后端实现功能的侧重点

本文实现的软件供应链成分分析工具基于 python 语言，使用了 Flask 库、argparse 库和 subprocess 库进行软件的开发。本文实现的软件成分分析工具源代码目录如图 5-4 所示，LibDetective_on_server 文件夹存储着命令行形式的 LibDetective 算法，是成分分析工具的算法核心。

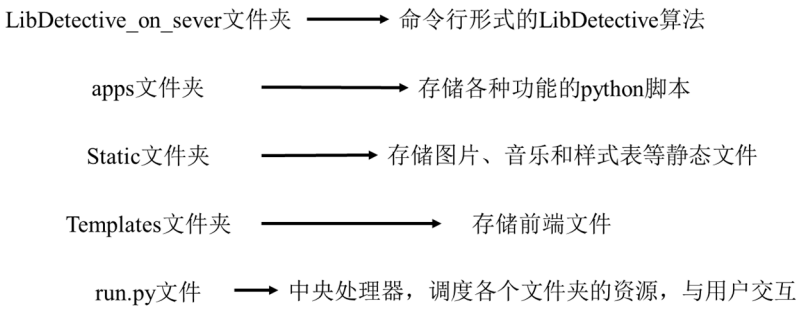


图 5-4　LibDetective 软件的目录和各文件的作用

在 python 的生态中，Flask 框架以模块的形式存在，只需要引入 Flask 库，即可基于该框架进行网站的开发。Flask 框架运行的主文件通常命名为 run.py，主文件的角色类似于一个中央处理器，后端中与用户进行交互的功能主要由主文件实现。服务器一般在云端，在服务器上运行软件主要靠命令行，因此需要利用 argparse 库对前面章节实现的 LibDetective 算法进行封装，实现命令行调用 LibDetetctive 算法。argparse 库具有支持各种类型的参数解析和能够生成参数文档的功能，使用 argparse 库能定制 LibDetective 算法的输入参数并进行封装，在此基础上，根据用户的输入构造指令，调用 subprocess 库的 run 函数来运行 LibDetective 算法，LibDetective 算法的返回值是一个文本文件，该文件存储与上传的开源软件中匹配成功的候选函数的名称和对应的开源软件名称，run.py 文件将解析得到的文件返回浏览器端。run.py 负责客户和服务器的交互，是其他程序脚本的调用者，异常处理

模块主要存在于主文件中。

5.3 性能测试

Centri[43]框架本身不能实现函数匹配功能。相反的，LibDetecive 使用了区域匹配技术，能够实现函数匹配功能，经过人工比对后，可以证明函数匹配的准确性。由此得出，LibDetective 框架功能更齐全。

5.3.1 数据集选择

本文选择了三个来自 LibAM[50]框架的数据集。由于 LibAM[50]是基于二进制进行成分分析的，LibDetective 和 Centri[43]是基于源代码进行成分分析，一些二进制软件存在无法获取源码的问题，本文对原始数据集进行了轻微的改动。同时，本文在每个数据集中加入了部分干扰条目，以检测是否存在误报问题。干扰条目是未被重用第三方库，称为负样本，被重用的第三方库称为正样本。常见重用关系类型有 4 种，主要包括两种简单的重用关系和两种复杂的重用关系，重用关系的复杂度是影响检测效果的重要因素，从 LibAM[50]中引入的数据集分别侧重于检测不同的重用类型。数据集 1 中，目标库与候选库之间的重用大部分属于简单重用，该数据集侧重于检测 LibDetective 和 Centris[43]在处理简单重用关系时的效果。数据集 2 和数据集 3 中，目标库与候选库之间的重用大部分属于复杂重用，这两个数据集侧重于检测 LibDetective 和 Centris[43]在处理复杂重用关系时的效果。

表 5-4　3 个数据集的内容

目标库名称	Lzbench	TurboBench	precomp-cpp
数据集中库的数量	18	17	12
目标库重用的库的数量	16	15	9
干扰条目的数量	2	2	3

5.3.2 指标计算

本文利用上个小节中选择的 3 个数据集，对 Centris[43]和 LibDetective 在每个数据集上分别进行了一次实验。实验的步骤如图 5-5 所示，将目标软件和候选开源软件分别输入 Centris[43]框架和 LibDetective 框架，两个框架会按照各自的算法对软件进行处理、解析和检测，最后输出结果。基准真相中注明了目标软件具体重用的候选开源软件类型，将结果与基准真相进行对比，统计出现漏报、误报等情况库的数量，分别计算两种框架的准确率、精确率和召回率，即可完成二者性能的比较。

准确率（Accuracy）、精确率（Precision）和召回率（Recall）的计算方法如式（5-1）、式（5-2）和式（5-3）所示。其中，真阴性（True Negative）表示没有检测到负样本；真阳性（True Positive）表示检测到正样本；假阴性（False Negative）表示本来是正样本，但是未被检测到，又称“漏报”；假阳性（False Positive）表示本来是负样本，但是被检测到了，又称“误报”。

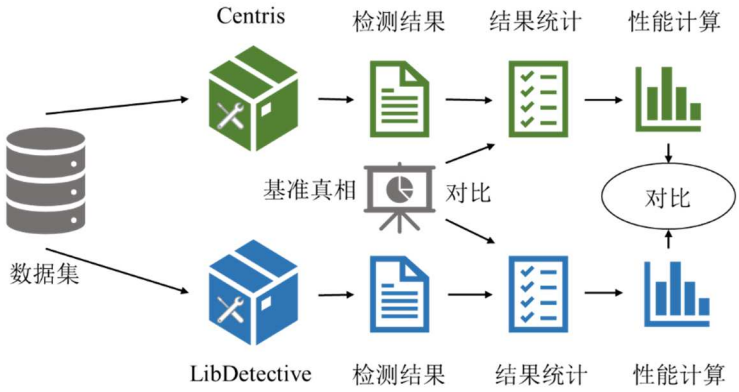


图 5-5　利用数据集对 LibDetective 和 Centris 框架进行测试的流程

$$Acc=\frac{TP+TF}{TP+TN+FP+FN} \tag{5-1}$$

$$Pre=\frac{TP}{TP+FP} \tag{5-2}$$

$$Recall=\frac{TP}{TP+FN} \tag{5-3}$$

经过三个数据集的测试，统计两种框架在三个数据集上的检测情况，统计出二者的真阳性、真阴性、假阳性、假阴性的数量如表 5-5 所示。计算出二者的准确率、精确率和召回率，结果如图 5-6、图 5-7、图 5-8 和表 5-6 所示。

表 5-5　Centris 和 LibDetective 在 3 个数据集上的检测情况统计

数据集	数据集 1		数据集 2		数据集 3	
框架	Centris	LibDetective	Centris	LibDetective	Centris	LibDetective
真阳性（TP）	10	13	2	11	3	7
真阴性（TN）	1	0	2	1	3	3
假阳性（FP）	0	1	0	1	0	0
假阴性（FN）	6	3	13	4	6	2
总计	17	17	17	17	12	12

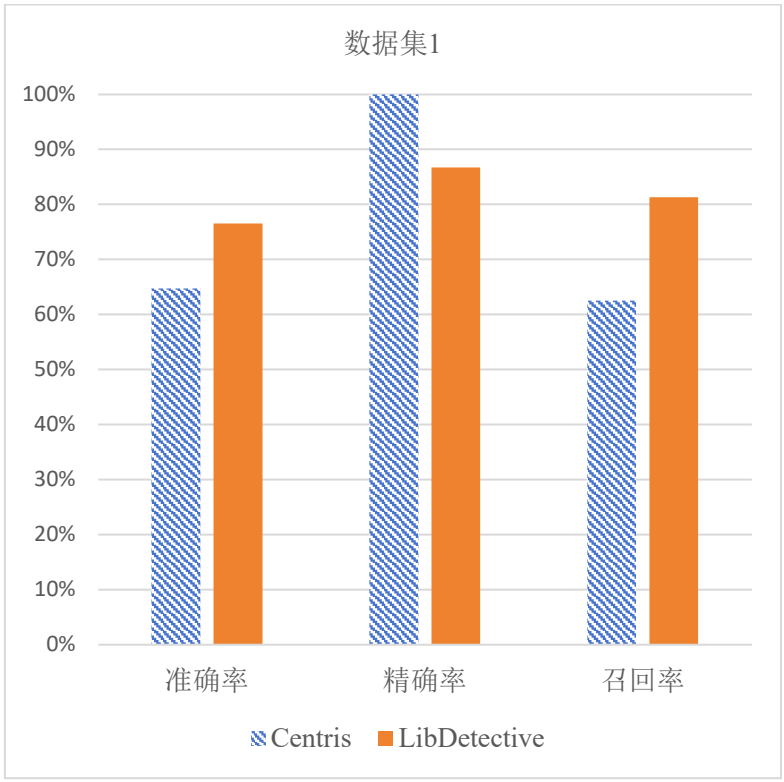


图 5-6　Centris 和 LibDetective 在数据集 1 上的准确率、精确率和召回率对比

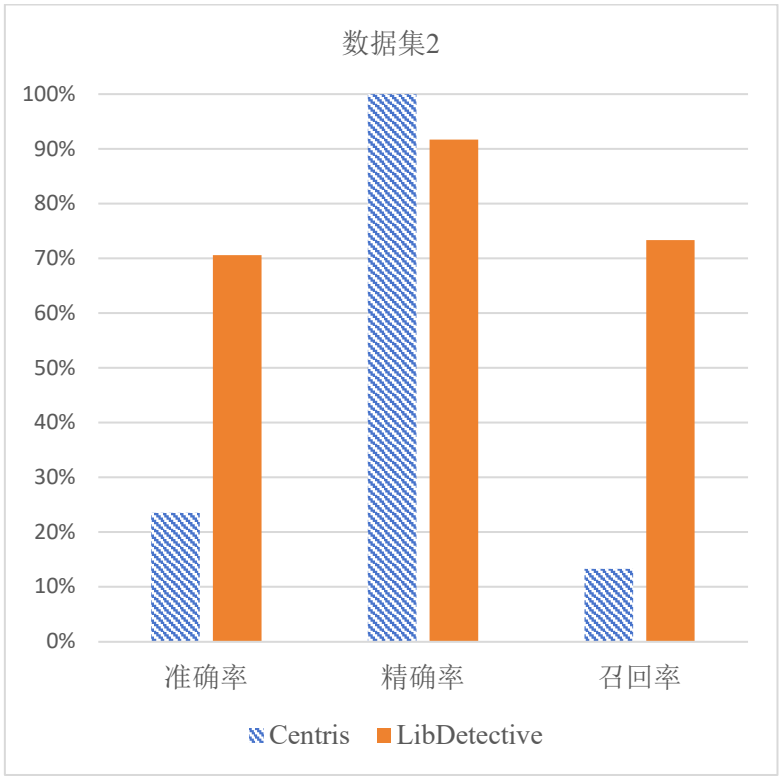


图 5-7　Centris 和 LibDetective 在数据集 2 上的准确率、精确率和召回率对比

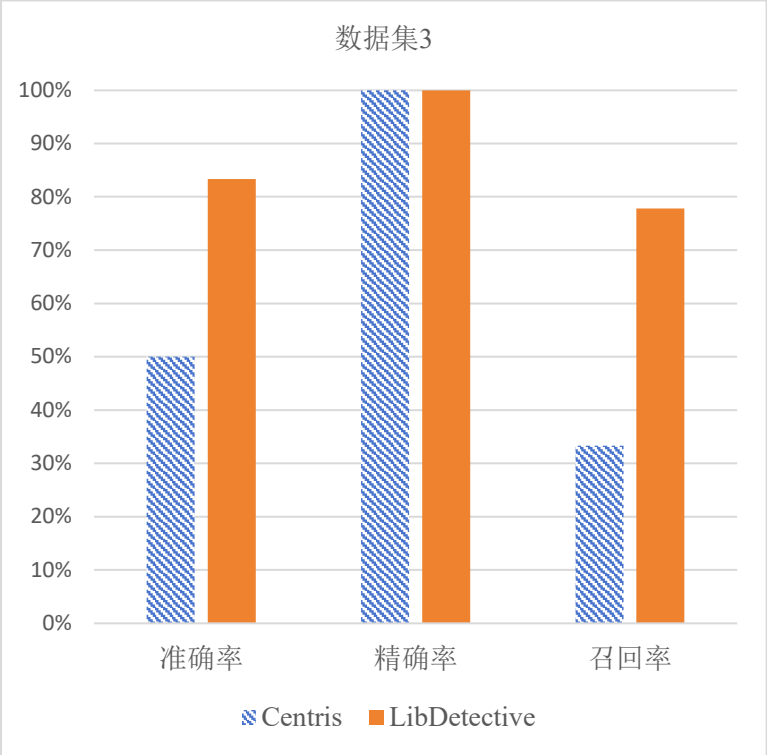


图 5-8　Centris 和 LibDetective 在数据集 3 上的准确率、精确率和召回率对比

表 5-6　Centris 和 LibDetective 在 3 个数据集上的检测指标统计

数据集	数据集 1		数据集 2		数据集 3	
框架	Centris	LibDetective	Centris	LibDetective	Centris	LibDetective
准确率	0.647	0.765	0.235	0.706	0.5	0.833
精确率	1	0.867	1	0.917	1	1
召回率	0.625	0.813	0.133	0.733	0.333	0.778

在三组数据集中，LibDetective 框架的性能与 Centris[43]框架的性能相比有了不同程度的提升，准确率和召回率在 3 个数据集上提高的百分比如表 5-7 所示。两种框架在三组数据集中表现出的精确率都很高，但是准确率和召回率有较大的差距。在侧重于简单重用关系检测的数据集 1 的测试中，两种框架的检测效果都较好，但 LibDetective 仍然表现出更好的性能，与 Centris[43]相比，准确率提高了 12%，召回率提高了 19%；在侧重于复杂重用情况检测的数据集 2、数据集 3 的测试中，LibDetective 的性能与 Centris[43]相比有了质的提高，证明了 LibDetective 具有处理复杂重用关系的能力。在数据集 2 的测试中，LibDetective 的准确率是 Centris[43]的接近 3 倍，提高了 47%，召回率是 Centris[43]的 5.5 倍，提高了 60%；在数据集 3 的测试中，LibDetective 框架的准确率是 Centris[43]框架准确率的接近 1.7 倍，提高了 33%，召回率是 Centri[43]的 2.3 倍，提高了 44%，这说明 LibDetective 框架能够

处理复杂的重用关系。

表 5-7　LibDetective 框架相比于 Centris 框架在 3 个数据集上性能提高的百分比

	准确率提高的百分比	召回率提高的百分比
数据集 1	12%	19%
数据集 2	47%	60%
数据集 3	33%	44%

LibDetective 的匹配方法引入了区域匹配技术，将分开的函数连接成区域进行匹配，同时，LibDetective 利用本文提出的基于节点的度的区域相似度比较方法，使得检测准确率显著提升。优点明显的同时，LibDetective 框架也存在一些缺点，典型的是依然存在少量的误报问题。经过对 LibDetective 框架匹配函数结果的分析，发现误报的库与目标库匹配成功的函数通常很少，而且内容相差很大，可以很轻松的人工排除。第三方库检测可以很好的辅助漏洞关联，而漏洞关联也可以提高漏洞修复的效率。因此漏报其实比误报更为严重，漏报意味着能修复这些漏洞的可能性变小，误报意味着后续阶段工作量的增加，但漏洞是还是会被修复的。

5.4　本章小结

本章经过对项目应用场景的分析，选择构建基于浏览器/服务器框架的软件，并选择 Apache 作为服务器软件，Flask 作为 Web 应用框架，综合分析了软件前后端面临的任务需求，完成了前后端代码的编写，实现了软件前端与后端的接口和功能，同时设计了命令行调用形式的 LibDetective 算法，完成了开发成分分析工具的任务。本文实现的软件成分分析工具前端基于 html5 语言，实现软件用户界面的构建；后端基于 python 语言，引入了 subprocess 模块、argparse 模块和 Flask 模块，实现了接收输入、解析输入和返回输出的功能，并具备一定的异常处理能力。经测试，该网站能够快速地检测用户输入并根据用户输入返回解析结果或对应的提示信息，具有实用性。

在实现成分分析工具的基础上，本章从 LibAM[50]中选取了三组数据集，对数据集中无法获取源代码的条目进行了删除和替换，并加入了部分干扰条目，形成三个最终数据集，三个最终数据集分别侧重于检测不同的重用关系，数据集 1 中目标软件与开源软件之间多为简单重用关系，该数据集侧重于检测简单重用关系；数据集 2 和数据集 3 中目标软件与开源软件之间多为复杂重用关系，该数据集侧重于检测复杂重用关系。利用最终数据集对 LibDetective 框架和 Centris[43]框架进行测试，得到检测结果并与基准真相进行对比，统计检测正确和检测错误的条目数量

得到最终结果，利用该结果进行准确率、精确率和召回率的计算，发现二者检测的精确率均很高，但是 LibDetective 框架与 Centris[43]框架相比在准确率和召回率上有了明显的提升。特别是在侧重于检测复杂重用关系的数据集中，Centris[43]的漏报现象严重，检测效果不佳，LibDetective 仍能保持较好的性能，在这两个数据集中，LibDetective 框架的准确率相比 Centris[43]分别提高了 47%和 33%，召回率分别提高了证明 60%和 44%，证明 LibDetective 算法能够检测复杂重用关系。同时，由于 LibDetective 框架能返回重用的函数清单，而 Centris[43]只能返回重用开源软件的类型，证明了 LibDetective 功能的多样性。LibDetective 的良好表现，有力的说明区域匹配技术在输入为源代码的情况下也存在较好的适用性，后续研究可以继续引入区域匹配技术，使用其他算法替换本文所使用的算法，实现软件成分分析算法对应子阶段的任务，完成软件成分分析的流程。

结　论

本文对软件组成分析技术的流程进行了研究，利用合适的方法实现了流程的各个环节，最终提出并实现了对开源软件进行组成分析的 LibDetective 算法。在实现算法的基础上，本文分析了项目的应用场景，选择了合适方法，开发了一款软件成分分析工具。最后，选用和改造了多组数据集，对 LibDetective 和当前主流测试框架 Centris 进行了对比实验，证明了 LibDetective 具有良好的性能和相对完备的功能，同时，也验证了在输入为源代码的情况下区域匹配技术依然能够保持优越性。

本文的具体工作归纳如下：

（1）提出了基于节点的度一种图嵌入方法，该方法还具有可扩展性强、可解释性强的特点。经过测试，验证了该方法的有效性和准确性。

（2）设计并实现了一款软件成分分析工具，LibDetective。该工具引入了区域匹配技术，同时提出和使用了基于节点的度的一种图嵌入方法，实现了对库和函数的匹配。经过测试，证明其有良好的性能。同时，区域匹配技术在输入为源代码的情况下具有优越性的猜测也得到了验证。

（3）搭建了一款以 LibDetective 为内核的成分分析工具。经过测试，验证软件可以正常运行。

LibDetective 框架的局限性如下：

（1）误报问题有待改善。

LibDetective 会存在少量的误报。经过分析，作者认为误报的原因来自：

a. 静态分析工具本身的误差。本文调用了 ctags 工具和 cflow 工具对程序源码进行静态分析，这些工具并不是完全准确的，只是误差比较小。将有误差的解析结果输入下一个环节，会对检测结果造成负面影响。

b. 图嵌入算法还可以继续改进。在后续的工作中，可以对现有的图神经网络进行改良和迁移，同时自己构造数据集来训练网络；也可以自己设计一个神经网络来完成目的。对于本文提出的基于节点的度的图嵌入方法，也可以改进，例如，根据图中节点深度的不同进行加权等，也是可以尝试和测验的方法。

c. 阈值可以动态调整。进行函数对匹配时，采用了余弦相似度与阈值进行对比的方法，阈值不同，测验结果会有很大的不同。固定阈值显然不是一种很好的办法，因此，可以构思一种动态调整阈值的方法，并进行测试验证。

（2）检测速度还有提升的空间。

作者认为延缓检测速度的原因主要有两点：

a. 频繁的调用外部工具解析目标代码，降低了程序的运行效率。

b. 在遍历操作中，很多相同的操作可能被运行了很多次，花费了一些重复的时间。之后可以通过引入多线程技术或者优化算法等，提高程序的运行速度。

（3）可以引入深度学习技术实现 LibDetective 算法的功能。

a. 设计和使用图神经网络实现图嵌入。在后续的学习中，可以多学习图神经网络的理论，自己设计一个能满足本项目任务需求的图神经网络，只需要替换 LibDetective 中的图嵌入环节，其他环节均不改变，形成新的框架，在进行测试验证。

b. 在生成数据库阶段，利用 Transformer 结构的编码器对源代码进行处理，生成特征向量，替换原有的局部敏感哈希算法。这个思路原理上是可行的，可以实现后进行测试，验证效果。

参考文献

[1] Geeksforgeeks. Software and its Types[EB/OL].(2023-08-29)[2024-05-10]. https://www.geeksforgeeks.org/software-and-its-types/.

[2] O'REILLY T. Lessons from open-source software development[J]. Communications of the ACM, 1999, 42(4): 32-37.

[3] KROGH G. Open-source software development[J]. MIT Sloan Management Review, 2003, 44(3): 14-18.

[4] HÖST M, ORUČEVIĆ-ALAGIĆ A. A systematic review of research on open source software in commercial software product development[J]. Information and Software Technology, 2011, 53(6): 616-624.

[5] GONZALEZ-BARAHONA J. A brief history of free, open source software and its communities[J]. Computer, 2021, 54(2): 75-79.

[6] MARACKE C. Free and Open Source Software and FRAND-based patent licenses: How to mediate between Standard Essential Patent and Free and Open Source Software[J]. The Journal of World Intellectual Property, 2019, 22(3-4): 78-102.

[7] RAGHUNATHAN S, PRASAD A, MISHRA B, CHANG H. Open source versus closed source: software quality in monopoly and competitive markets[J]. IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans, 2005, 35(6): 903-918.

[8] Wayback Machine. GITHUB USRS[EB/OL].(2022-04-25)[2024-05-10].https://web.archive.org/web/20220425042112/https://github.com/search?q=type:user&type=Users.

[9] Github. Celebrating nine years of GitHub with an anniversary sale [EB/OL].(2017-04-10)[2024-05-10].https://github.blog/2017-04-10-celebrating-nine-years-of-github-with-an-anniversary-sale/.

[10] GOUSIOS G, VASILESCU B, SEREBRENIK A. Lean GHTorrent: GitHub data on demand[C]//Proceedings of the 11th Working Conference on Mining Software Repositories. Hyderabad, India, 2014: 384-387.

[11] Wayback Machine.100 million developers and counting [EB/OL].(2023-01-25) [2024-05-10]. https://web.archive.org/web/20220425042112/https://github.com/search?q=type:user&type=Users.

[12] BONACCORSI A, ROSSI C. Why open source software can succeed[J]. Research Policy, 2003, 32(7): 1243-1258.

[13] BANDYOPADHYAY S, THAKUR S. ICT in education: open source software and its impact on teachers and students[J]. International Journal of Computer

Applications, 2016, 151(6): 19-24.

[14] SEIDER D, LITZ M, SCHREIBER A. Open source software framework for applications in aeronautics and space[C]//2012 IEEE Aerospace Conference. Big Sky, Montana, USA , IEEE, 2012: 1-11.

[15] SHOULING J, QINYING W, ANYING C. Survey on Open-source Software Supply Chain Security[J]. Journal of Software, 2022, 34(3): 1330-1364.

[16] GEER D, TOZER B, MEYERS J. For good measure: Counting broken links: A quant's view of software supply chain security[J]. USENIX, 2020, 45(4):83-86.

[17] Law Insider. Third Party Software Components definition[EB/OL].(2024-5-10)[2024-5-10].https://www.lawinsider.com/dictionary/third-party-software-components.

[18] 徐俊 ，胡蓉 ，余镭 ．关于软件供应链安全的网络安全保险研究 [J]. 网络空间安全, 2022, 13(4): 62-69.

[19] PRANA G, SHARMA A, SHAR L. Out of sight, out of mind? How vulnerable dependencies affect open-source projects[J]. Empirical Software Engineering, 2021, 26: 1-34.

[20] SEJFIA A, SCHÄFER M. Practical automated detection of malicious npm packages[C]//Proceedings of the 44th International Conference on Software Engineering. Pittsburgh, Pennsylvania, 2022: 1681-1692.

[21] OHM M, PLATE H, SYKOSCH A. Backstabber's knife collection: A review of open source software supply chain attacks[C]//Detection of Intrusions and Malware, and Vulnerability Assessment: 17th International Conference, Lisbon, Portugal, 2020: 23-43.

[22] Welivesecurity. Analysis of TeleBots' cunning backdoo [EB/OL].(2017-07-04)[2024-05-10].https://www.welivesecurity.com/2017/07/04/analysis-of-telebots-cunning-backdoor/.

[23] The Register. How one developer just broke Node, Babel and thousands of projects in 11 lines of JavaScript[EB/OL].(2016-03-23)[2024-05-10]. https://www.theregister.com/2016/03/23/npm_left_pad_chaos/.

[24] dev. How a Rogue Developer Ruined Millions of Software (happened this weekend)[EB/OL].(2022-01-10)[2024-05-10].https://dev.to/anthonyjdella/how-a-rogue-developer-ruined-millions-of-software-happened-this-weekend-4bp.

[25] Forbes. Supply Chains Are In The Cyberattack Crosshairs [EB/OL].(2022-04-27)[2022-05-10].https://www.forbes.com/sites/forbestechcouncil/2022/04/27/supply-chains-are-in-the-cyberattack-crosshairs/?sh=72644fb11808

[26] Google Cloud. Highly Evasive Attacker Leverages SolarWinds Supply Chain to Compromise Multiple Global Victims With SUNBURST Backdoor [EB/

OL].(2020-12-13)[2024-05-10].https://cloud.google.com/blog/topics/threat-intelligence/evasive-attacker-leverages-solarwinds-supply-chain-compromises-with-sunburst-backdoor.

[27] RAYMOND E. The cathedral and the bazaar[J]. Knowledge, Technology & Policy, 1999, 12(3): 23-49.

[28] TECHMONITER. 2022 Open Source Security and Risk Analysis Report [R/OL].(2020-12-13)[2024-05-10].https://techmonitor.ai/whitepapers/2022-open-source-security-and-risk-analysis-report.

[29] 奇安信. 2023 中国软件供应链安全分析报告[R/OL].(2024-05-10)[2024-05-10].https://www.qianxin.com/threat/reportdetail?report_id=297.

[30] DURUMERIC Z, LI F, KASTEN J. The matter of heartbleed[C]//Proceedings of the 2014 Conference on Internet Measurement Conference. Vancouver, BC, Canada, 2014: 475-488.

[31] HIESGEN R, NAWROCKI M, SCHMIDT T. The race to the vulnerable: Measuring the log4j shell incident[J]. ArXiv Preprint ArXiv, 2022.

[32] CISA. Securing the Software Supply Chain: Recommended Practices for Developers[R/OL]. (2022-08)[2024-05]. https://www.cisa.gov/sites/default/files/publications/ESF_SECURING_THE_SOFTWARE_SUPPLY_CHAIN_DEVELOPERS.PDF.

[33] SCHREIBER A, GALOPPINI R, MEINEL M. An open source software directory for aeronautics and space[C]//Proceedings of The International Symposium on Open Collaboration. Berlin, Germany, 2014: 1-7.

[34] nasa. nasa code nasa gov[EB/OL]. (2024-05-10)[2024-05-10]. https://code.nasa.gov/.

[35] ZDNet. SpaceX: We've launched 32,000 Linux computers into space for Starlink internet[EB/OL]. (2020-06-08)[2024-05-10]. https://www.zdnet.com/article/spacex-weve-launched-32000-linux-computers-into-space-for-starlink-internet/.

[36] hackaday. the usage of embedded linux in spacecraft [EB/OL]. (2024-02-10)[2024-02-10]. https://hackaday.com/2024/02/10/the-usage-of-embedded-linux-in-spacecraft/

[37] Matrod. industries aerospace[EB/OL]. (2024-5-10)[2024-5-10]. https://www.matroid.com/industries/aerospace.

[38] PRANA G, SHARMA A, SHAR L. Out of sight, out of mind? How vulnerable dependencies affect open-source projects[J]. Empirical Software Engineering, 2021, 26: 1-34.

[39] 虹科网络安全.什么是软件成分分析（SCA）？[EB/OL]. (2023-11-27)[2024-5-10]. https://zhuanlan.wangan.com/p/11v7ba1719b6e52c.

[40] CHEN Y, SANTOSA A E, SHARMA A, et al. Automated identification of libraries from vulnerability data[C]//Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Software Engineering in Practice. Seoul, South Korea, 2020: 90-99.

[41] DENNIS K. [M] Building Secure Cars: Assuring the Automotive Software Development Lifecycle. Wiley Data and Cybersecurity, 2021, 91-110.

[42] LI S, DONG C, WANG Y. LibDI: A Direction Identification Framework for Detecting Complex Reuse Relationships in Binaries[C]//IEEE Military Communications Conference. Boston,MA,United States, 2023: 741-746.

[43] WOO S, PARK S, KIM S. CENTRIS: A precise and scalable approach for identifying modified open-source software reuse[C]//International Conference on Software Engineering (ICSE). Madrid, Spain, 2021: 860-872.

[44] JIANG L, YUAN H, TANG Q. Third-Party Library Dependency for Large-Scale SCA in the C/C++ Ecosystem: How Far Are We?[C]//ACM SIGSOFT International Symposium on Software Testing and Analysis. Seattle, Washington, United States, 2023: 1383-1395.

[45] DUAN R, BIJLANI A, XU M. Identifying open-source license violation and 1-day security risk at large scale[C]//ACM SIGSAC Conference on Computer and Communications Security. Dallas, Texas, USA, 2017: 2169-2185.

[46] YUAN Z, FENG M, LI F. B2sfinder: Detecting open-source software reuse in cots software[C]//International Conference on Automated Software Engineering. San Diego, California, USA, 2019: 1038-1049.

[47] ZHAO B, JI S, XU J. A large-scale empirical analysis of the vulnerabilities introduced by third-party components in IoT firmware[C]//ACM SIGSOFT International Symposium on Software Testing and Analysis. Online, 2022: 442-454.

[48] TANG W, WANG Y, ZHANG H. Libdb: An effective and efficient framework for detecting third-party libraries in binaries[C]//International Conference on Mining Software Repositories. Pittsburgh, PA, USA, 2022: 423-434.

[49] WU J, XU Z, TANG W. Ossfp: Precise and scalable c/c++ third-party library detection using fingerprinting functions[C]//International Conference on Software Engineering (ICSE). IEEE, Melbourne, Australia, 2023: 270-282.

[50] LI S, WANG Y, DONG C. Libam: An area matching framework for detecting third-party libraries in binaries[J]. ACM Transactions on Software Engineering and Methodology, 2023, 33(2): 1-35.

[51] JIANG L, AN J, HUANG H. BinaryAI: Binary Software Composition Analysis via Intelligent Binary Source Code Matching[C]//International Conference on Software Engineering. Lisbon, Portugal, 2024: 1-13.

[52] OLIVER J, CHENG C, CHEN Y. TLSH--a locality sensitive hash[C]// Fourth Cybercrime and Trustworthy Computing Workshop. Sydney NSW, Australia, 2013: 7-13.

[53] XU M. Understanding graph embedding methods and their applications[J]. SIAM Review, 2021, 63(4): 825-853.

[54] VISHWANATHAN S, SCHRAUDOLPH N, KONDOR R. Graph kernels[J]. The Journal of Machine Learning Research, 2010, 11: 1201-1242.

[55] LAN Z, HONG B, MA Y. More interpretable graph similarity computation via maximum common subgraph inference[J]. IEEE Transactions on Knowledge and Data Engineering, 2024, 14(8): 1-14.

[56] ZHOU J, CUI G, HU S. Graph neural networks: A review of methods and applications[J]. AI Open, 2020, 1: 57-81.

[57] STECK H, EKANADHAM C, KALLUS N. Is cosine-similarity of embeddings really about similarity?[C]//ACM on Web Conference 2024. Singapore, Singapore, 2024: 887-890.

哈尔滨工业大学本科毕业论文（设计）

原创性声明和使用权限

本科毕业论文（设计）原创性声明

本人郑重声明：此处所提交的本科毕业论文（设计）《基于区域匹配技术的软件供应链成分分析工具设计与实现》，是本人在导师指导下，在哈尔滨工业大学攻读学士学位期间独立进行研究工作所取得的成果，且毕业论文（设计）中除已标注引用文献的部分外不包含他人完成或已发表的研究成果。对本毕业论文（设计）的研究工作做出重要贡献的个人和集体，均已在文中以明确方式注明。

作者签名：　　日期：　　年　　月　　日

本科毕业论文（设计）使用权限

本科毕业论文（设计）是本科生在哈尔滨工业大学攻读学士学位期间完成的成果，知识产权归属哈尔滨工业大学。本科毕业论文（设计）的使用权限如下：

（1）学校可以采用影印、缩印或其他复制手段保存本科生上交的毕业论文（设计），并向有关部门报送本科毕业论文（设计）；（2）根据需要，学校可以将本科毕业论文（设计）部分或全部内容编入有关数据库进行检索和提供相应阅览服务；（3）本科生毕业后发表与此毕业论文（设计）研究成果相关的学术论文和其他成果时，应征得导师同意，且第一署名单位为哈尔滨工业大学。

保密论文在保密期内遵守有关保密规定，解密后适用于此使用权限规定。

本人知悉本科毕业论文（设计）的使用权限，并将遵守有关规定。

作者签名：　　日期：　　年　　月　　日

导师签名：　　日期：　　年　　月　　日

致　谢

感谢我的父母。父母永远是我坚强的后盾，高考结束以后，母亲时常自嘲：自己成了空巢老人。哈尔滨到泸州 3000 公里，北京到泸州 2000 公里，无论距离多遥远，背后爸爸妈妈的支持，将使我走得更加稳健。在这里，真诚的向父母道一声感谢，您们辛苦了，我已长大，请放心。

感谢我的导师：刘延芳老师和李红老师。刘延芳老师认真负责，一丝不苟，虽然非常事务繁忙，但依旧在学生身上倾注了大量的时间与精力。刘老师非常关心我，在保研结果公布之前的很长一段时间里，为我专业选择、参加竞赛和专业课的学习等出谋划策。我的保研之路并不是一帆风顺，但刘老师的帮助让这条路平坦了许多。除此之外，我能来所里做毕设，刘老师承担了很大的责任和风险，签了很多字，谈了很多话，刘老师毫无怨言，还表达了对我的全力支持。对刘老师万分的感激和感动，我会铭记在心里，在这个特殊的地方，汇聚成一句看似漫不经心但是发自内心的话：感谢刘老师，您辛苦了。李红老师严谨随和，我转专业以后，李老师单独指导我，为我指明前行的方向。您推荐的书和课程，由于目前我的能力和精力有限，暂时不能完全理解和消化，但已经受益匪浅。以后的学习生涯中，还需要李老师的帮助与指导。

感谢我的师兄李思远。师兄年轻有为，耐心认真，在我毕业设计的各个方面，给我提建议，想方法。虽然我能力有限，但师兄总是耐心的指导我和鼓励我，让我充满信心的面对困难与挑战，我的毕业设计得以顺利完成。师兄即将启程留洋，祝师兄万事顺利，前程似锦。

感谢我的大学期间舍友们。和你们在一起，我感到幸运与快乐。过去的四年，给我留下来太多美好的回忆。这四年是和睦的，也是收获满满的。我希望我们可以一直做可以无话不说的朋友，可以直言不讳的朋友。

感谢我最亲爱的朋友小徐，你是我的开心果，是我前进的动力。感谢你给予我的各方面的支持与帮助，希望我们能一直做彼此坚强的后盾。